Sustainable Smart City Project Using IBM Granite LLM

This document presents a comprehensive overview of the Sustainable Smart City project, which leverages the IBM Granite Large Language Model (LLM) to enhance urban sustainability, efficiency, and citizen engagement. It covers the project's purpose, architecture, setup, features, user interface, API, testing, known issues, and future enhancements. The goal is to provide city administrators, planners, and citizens with intelligent tools to optimize resource use and promote sustainable urban development.

TEAM DETAILS

Team ID : NM2025TMID03719

Team Size : 4

Team Leader : DEVADHARSHINI S

Team member : DHANALAKSHMI R

Team member : JANANI A

Team member : KARTHIGA S

Project Overview and Purpose

- The Sustainable Smart City project is an ambitious initiative aimed at developing an advanced smart urban assistant that promotes ecological sustainability and efficient city management. By harnessing the power of IBM Granite Large Language Model (LLM), this system is designed to intelligently analyze vast and diverse city data, enabling it to answer complex queries and provide actionable insights. It focuses on critical urban challenges such as energy consumption, waste management, traffic control, and pollution reduction, thereby fostering a more sustainable and livable urban environment for all residents.
- The project’s essential goal is to empower both city decision-makers and citizens by providing them with the tools they need to actively engage in sustainability efforts. By leveraging real-time and historical data from various city sensors and public data sources, the system offers context-aware, accurate responses to inquiries that range from energy usage patterns to waste disposal recommendations. This not only aids in informed decision-making but also encourages proactive community participation in sustainability initiatives.
- Key features of the system include intelligent query handling, which ensures answers are tailored to the user's context and the latest available data, enhancing usability across diverse user profiles. The provision of real-time data insights enables dynamic monitoring and management of city conditions, while the system’s recommendations for optimizing energy consumption and resource allocation help reduce operational costs and environmental impact. Furthermore, the system generates automated, detailed sustainability reports that track progress and compliance with environmental goals, serving as a critical resource for city planners and policy makers.
- To maximize accessibility, the project incorporates an interactive chat interface designed with user-friendliness in mind, making it approachable for users with different levels of technical expertise4from expert analysts to everyday citizens. This interface allows seamless interaction with the system, thereby democratizing access to sustainability-related information and fostering wider engagement.

Project Folder Structure

The project is organized into a clear folder structure to facilitate development and maintenance. The root directory contains the main application entry point and dependency files. Subdirectories include configuration files, local models, stored data and databases, API endpoint implementations, frontend code, utility functions, automated tests, log files, and documentation.

This modular organization supports scalability and ease of navigation for developers and administrators managing the system.

- `app.py` – Main application entry point
- `requirements.txt` – Python dependencies
- `config/` – Configuration files
- `models/` – Local AI models if needed
- `data/` – Stored data and database files
- `api/` – API endpoint implementations
- `frontend/` – Frontend code (React/Vue)
- `utils/` – Helper functions and utilities
- `tests/` – Automated test cases
- `logs/` – Log files for debugging
- `docs/` – Documentation files

Running the Application

To launch the application, simply run the command `python app.py` in your terminal or command prompt. This will start a local web server that you can access by navigating to http://localhost:8000 in your web browser.

Once the server is running, you can interact with the system in two ways - through the user-friendly web interface, or by sending API requests directly. The web interface provides an intuitive way for users to access the system's features and functionality, while the API allows for programmatic integration with other systems or applications.

For production environments, where the system needs to handle a larger number of concurrent users and maintain high availability, it is recommended to deploy the application using a containerization platform like Docker. This will ensure that the system is scalable, reliable, and easy to manage and maintain. Alternatively, the application can be deployed on a cloud platform, which will provide additional benefits such as automatic scaling, load balancing, and backup/recovery capabilities. Regardless of the deployment method, the system is designed to support multiple users interacting with the application simultaneously, as well as automated report generation and other scheduled tasks. This ensures that the smart city platform remains responsive and efficient, even as the user base and data volume grows over time.

API Documentation

The system exposes Restful API endpoints to facilitate interaction and integration. The primary endpoints include:

- **POST /api/query:** Submit sustainability-related queries. The request body contains a JSON object with a "query" field. The response returns context-aware answers generated by the IBM Granite LLM.
- **GET /api/reports:** Retrieve the latest sustainability report, including summary data such as energy usage reductions and other key metrics.

Example request for query submission:

```
{
"query": "How to reduce city energy consumption?"
}
```

Authentication and Security

The system employs a robust authentication and security model to protect sensitive data and ensure controlled access to the application's functionality. All requests to the /api/* endpoints are required to include a API_KEY header, which serves as the primary authentication mechanism for the system's valid API.

In addition to the API key-based authentication, the frontend user sessions are managed using JSON Web Tokens (JWT). This allows for secure and persistent user authentication, ensuring that users can seamlessly navigate the application without having to repeatedly provide their credentials.

For administrative users, the system provides elevated privileges that grant them the ability to manage critical aspects of the application, such as data sources, user accounts, and system configurations.These administrative functions are accessible through additional secured endpoints, which are designed to protect sensitive information and prevent unauthorized modifications.

This layered security approach, combining API key-based authentication, JWT-based user sessions, and role-based access controls, ensures that the system maintains a high level of security and data protection. By implementing these robust security measures, the application can safely handle sensitive data and provide a secure environment for users to interact with the smart city platform.

User Interface Features

The system's web interface is designed with a clean, modern aesthetic that provides an intuitive and responsive user experience, optimized for both desktop and mobile devices. The interface leverages a minimalist design approach, placing the focus squarely on the key functionality .

At the heart of the user experience is an advanced chatbot that utilizes natural language processing capabilities to accept a wide range of user queries related to urban sustainability. This chatbot acts as a conversational assistant, empowering users to explore the system's features and data through natural, conversational interactions. For administrative users, the system includes a dedicated admin panel that enables the management of critical system functions. This panel supports tasks such as user account administration, API key configuration, and the generation of detailed reports on the city's sustainability data and performance trends.

Testing of the Project

1. Unit Testing

- Test individual API endpoints to ensure correct input/output behavior.
- Validate database model creation, updates, and retrieval of city-related data (e.g., traffic records, energy reports).
- Test service layer methods, especially IBM Granite LLM integration logic, to ensure proper handling of input queries and responses.
- Verify utility functions (e.g., data preprocessing for IoT streams).

2. Integration Testing

- Test the complete flow from frontend query input API request IBM Granite LLM interaction ³ Response rendering on UI.
- Ensure correct data flow between database, backend logic, and frontend interface.
- Validate authentication workflows:

    User registration Login Role-based access.

    Token validation on protected endpoints.

- Test real-time IoT data ingestion pipeline and its effect on dashboards.

3. Performance Testing

- Measure API response time under normal and high query loads.
- Test system behavior with multiple concurrent users (e.g., simulate 100+ simultaneous queries).
- Analyze throughput and latency when processing large IoT data batches (e.g., thousands of data points per minute).

4. Security Testing

- Test JWT token generation and expiration behavior.
- Validate input sanitization to prevent injection attacks (e.g., SQL injection, command injection).

- Test API rate limiting to prevent abuse of IBM Granite LLM API.

5. UI/UX Testing

- Test responsiveness of the dashboard on various screen sizes (mobile, tablet, desktop).
- Validate usability of query input – whether natural language inputs yield correct and understandable responses.
- Ensure visual elements (graphs, maps, status indicators) update correctly with live data.

6. Regression Testing

- After code changes or enhancements, re-test critical workflows (querying, authentication, dashboard visualization) to prevent regression bugs.

7. Stress Testing

- Test system stability under extreme conditions (e.g., high-frequency IoT sensor data bursts or hundreds of parallel user queries).
- Verify system does not crash and handles failures gracefully (e.g., returns informative error responses).

8. Data Integrity Testing

- Ensure that historical IoT and user data are stored correctly in the database without corruption.
- Test data consistency after schema migrations or data updates.

9. Edge Case Testing

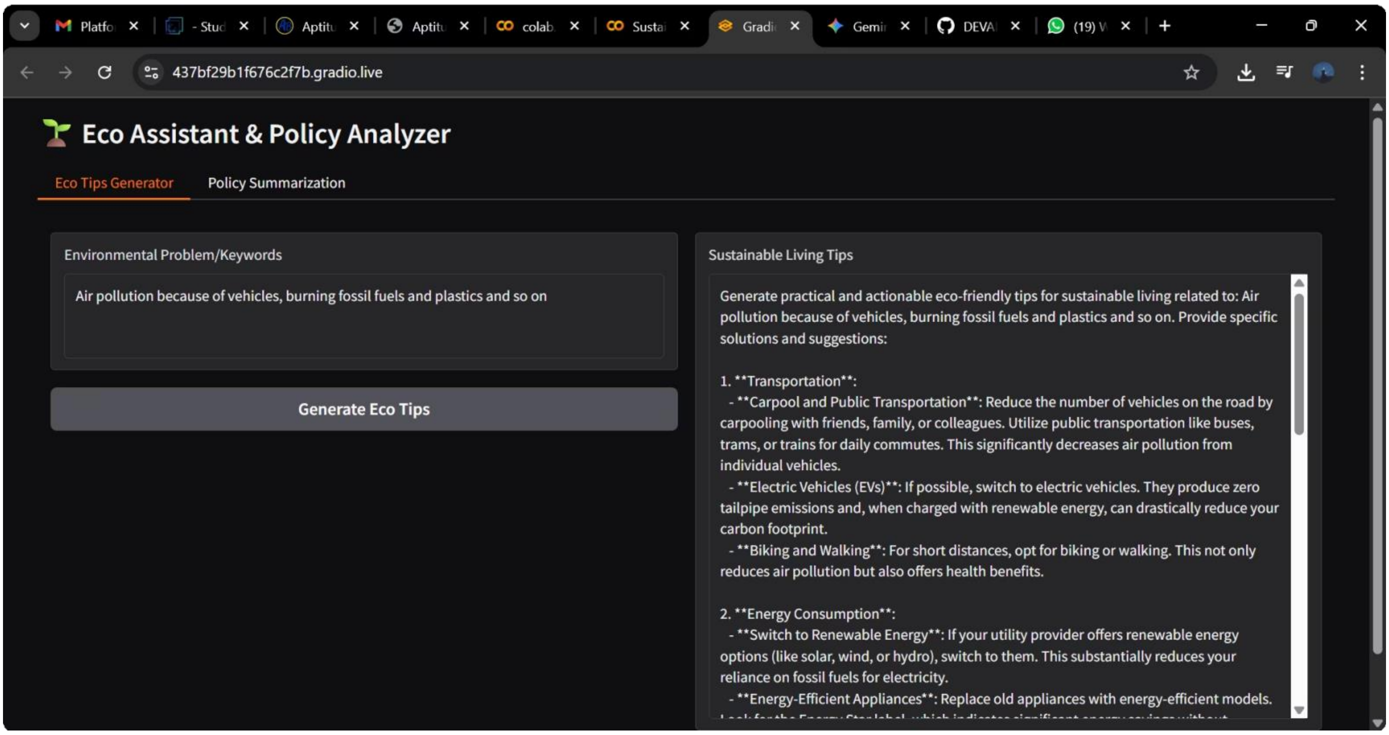
- Submit empty, malformed, or extremely long natural language queries to test LLM handling.
- Test system behavior when IoT sensors stop sending data or send incorrect values.

10. Automated Testing

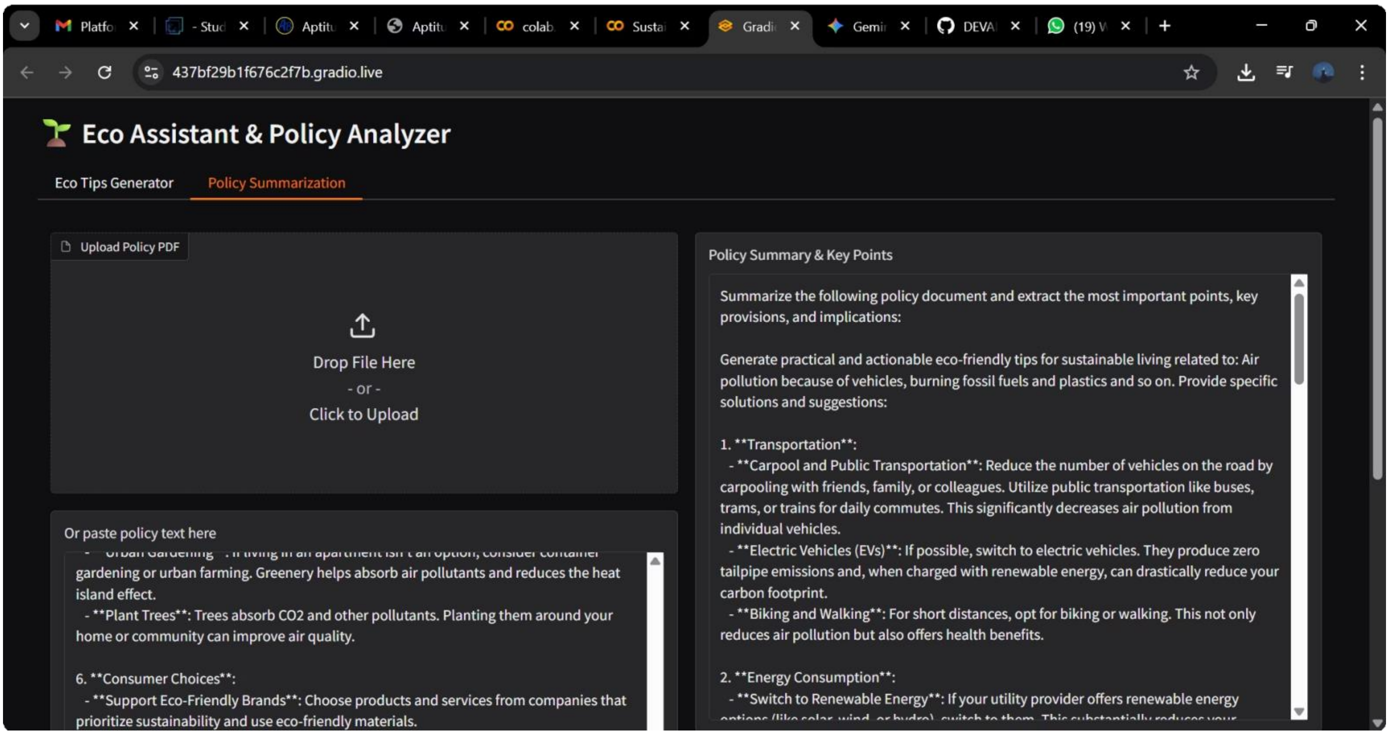
- Implement automated tests using frameworks like **pytest** for backend and **Jest** or **Cypress** for frontend.
- Set up a Continuous Integration (CI) pipeline to run tests automatically on every commit.

PROJECT OUTPUT

1. Provide any Environment related problems in the **Eco Tips Generator** Section & click on Generate Eco TIps. It will generate sustainable tips based on the given problem.



2. Give any policies or upload a policy pdf and click on **Generate Eco TIps**. It will automatically generate tips in the **Sustainable Living Tips** section based on the given policies.



Known Issues

- Delayed response times for complex or large-scale data queries due to external API rate limits.
- Inconsistent AI predictions when input data is incomplete or lacks sufficient context.
- User Interface (UI) responsiveness issues on smaller screen devices or low- resolution displays.
- Limited support for non-English languages; currently supports only English.
- Scalability constraints under heavy simultaneous user queries or high- frequency IoT data streams.
- Lack of real-time error handling when IoT sensors go offline or send corrupted data.
- Insufficient audit logs for user activities and system changes.
- Manual intervention needed for updating AI model knowledge base as new city infrastructure evolves.
- Security vulnerabilities if environment variables (e.g., API keys) are not properly secured.
- No offline mode available – application requires constant internet connectivity to function.
- Basic role-based access control with no fine-grained permission management. Limited historical
- data visualization features for long-term analysis.

Future Enhancements

- **Multilingual Query Support**
  Implement natural language query processing in multiple local languages (e.g., Hindi, Tamil, etc.) to improve accessibility for non-English speaking users.
- **Mobile Application Development**
  Build native mobile apps (Android and iOS) to provide citizens and city administrators on-the-go access to real-time insights and control over smart city services.
- **Advanced Predictive Analytics**
  Integrate advanced machine learning models to offer deeper predictive insights, such as energy consumption forecasts, traffic congestion predictions, and infrastructure maintenance needs.
- **Expanded IoT Device Integration**
  Support a broader range of IoT devices (e.g., noise pollution sensors, smart street lights, smart water meters) to collect more diverse real-time data streams.
- **Offline Data Access and Sync**
  Enable offline data access with local caching and sync capabilities for areas with unreliable internet connectivity.
- **Real-Time Anomaly Detection**
  Implement automated real-time anomaly detection for critical infrastructure (e.g., sudden spikes in energy usage, unexpected traffic congestion) with instant alerts.
- **Role-Based Fine-Grained Access Control**
  Enhance the authentication system with more advanced role-based access control (RBAC), allowing granular permissions for various administrative roles.
- **Automated Knowledge Base Updates**
  Develop automated pipelines to continuously update the AI model's knowledge base with new city regulations, infrastructure changes, and IoT device data.
- **Enhanced Security Features**
  Introduce additional security measures such as two-factor authentication (2FA), data encryption, and secure API key management practices.
- **Audit and Logging System**
  Implement a comprehensive logging and audit trail system to track user activities, system events, and model queries for accountability and debugging purposes.

Conclusion

The Sustainable Smart City project using IBM Granite LLM represents a significant step toward building intelligent, eco-friendly, and citizen-centric urban environments. By combining real-time IoT data, advanced AI-driven insights, and an interactive user interface, the system empowers both city administrators and citizens to make informed decisions that promote sustainability, efficiency, and community participation.

While the current implementation already provides valuable features such as intelligent query handling, sustainability reports, and an eco tips generator, the project also acknowledges challenges such as scalability, response delays, and limited multilingual support. Addressing these limitations through future enhancements—including predictive analytics, multilingual access, expanded IoT integration, and robust security—will further strengthen the platform's impact.

Ultimately, this project demonstrates how cutting-edge AI models like IBM Granite LLM can transform raw city data into actionable intelligence, creating smarter, greener, and more resilient cities. It lays the foundation for sustainable urban development where technology and community engagement work hand-in-hand to shape a better future.