**Sustainable Smart City Assistant -Project Documentation**

**Table of Contents**

1. Project Overview
2. System Architecture
3. Technology Stack
4. Project Structure
5. Implementation Details
6. Development Workflow
7. Setup and Installation
8. Features and Functionality
9. API Documentation
10. Screenshots and Results
11. Challenges and Solutions
12. Future Enhancements
13. Conclusion

**1. Project Overview**

**1.1 Introduction**

The **Sustainable Smart City Assistant** is an AI-powered platform designed to enhance urban governance and promote environmental sustainability. The platform integrates advanced AI technologies including IBM Watsonx Granite LLM and Pinecone vector database to provide intelligent solutions for city management, citizen engagement, and environmental monitoring.

**1.2 Objectives**

* **Primary Goal**: Create an intelligent assistant for sustainable urban development
* **Secondary Goals**:
  + Enable citizen feedback and engagement
  + Provide real-time environmental monitoring
  + Facilitate policy document management and search
  + Generate predictive analytics for city KPIs
  + Offer personalized eco-friendly recommendations

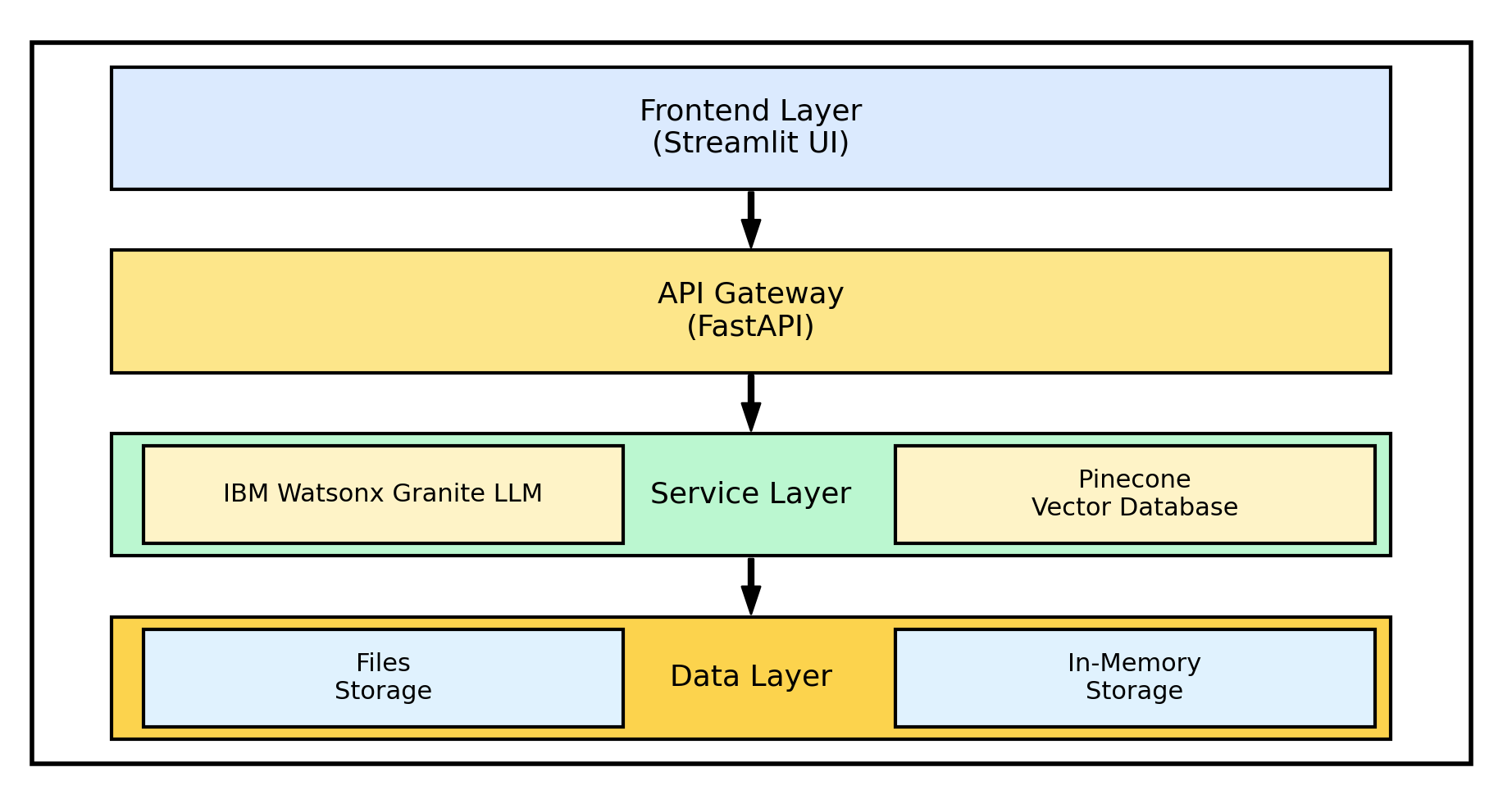
**1.3 Key Features**

* **Interactive Chat Assistant** powered by IBM Watsonx Granite LLM
* **Real-time KPI Dashboard** with sustainability metrics
* **Citizen Feedback System** for community engagement
* **Policy Document Search** using semantic vector search
* **Eco-friendly Tips Generator** for environmental awareness
* **Anomaly Detection** for unusual patterns in city data
* **Forecasting Models** for predictive analytics
* **Comprehensive Report Generation** for city officials

**2. System Architecture**

**2.1 Architecture Overview**

The system follows a **microservices architecture** with clear separation between frontend and backend components:



**2.2 Component Interaction Flow**

1. **User Input**: Users interact through Streamlit dashboard
2. **API Gateway**: FastAPI routes requests to appropriate services
3. **AI Processing**: IBM Watsonx processes natural language queries
4. **Vector Search**: Pinecone handles semantic document retrieval
5. **Data Analysis**: ML models process KPI data for insights
6. **Response**: Results rendered in user-friendly interface

**3. Technology Stack**

**3.1 Backend Technologies**

* **FastAPI**: Modern web framework for building APIs
* **Python 3.8+**: Core programming language
* **Pydantic**: Data validation and settings management
* **Uvicorn**: ASGI server for FastAPI

**3.2 Frontend Technologies**

* **Streamlit**: Interactive web application framework
* **Streamlit-Option-Menu**: Enhanced navigation components
* **Custom CSS**: Styling and UI enhancements

**3.3 AI and ML Technologies**

* **IBM Watsonx**: Enterprise-grade AI platform
* **Granite LLM**: Large language model for text generation
* **Sentence Transformers**: Text embedding generation
* **Scikit-learn**: Machine learning algorithms
* **Pandas**: Data manipulation and analysis

**3.4 Database and Storage**

* **Pinecone**: Vector database for semantic search
* **File System**: Local storage for documents and data
* **In-Memory Storage**: Session-based data management

**4. Project Structure**

**4.1 Directory Organization**

sustainable-smart-city-assistant/

├── app/ # Backend application

│ ├── \_\_init\_\_.py

│ ├── main.py # FastAPI main application

│ ├── api/ # API route handlers

│ │ ├── chat\_router.py

│ │ ├── feedback\_router.py

│ │ ├── eco\_tips\_router.py

│ │ ├── kpi\_upload\_router.py

│ │ ├── vector\_router.py

│ │ ├── policy\_router.py

│ │ └── dashboard\_router.py

│ ├── core/ # Core configurations

│ │ └── config.py

│ └── services/ # Business logic services

│ ├── granite\_llm.py

│ ├── pinecone\_client.py

│ ├── document\_embedder.py

│ ├── document\_retriever.py

│ ├── kpi\_file\_forecaster.py

│ └── anomaly\_file\_checker.py

├── frontend/ # Frontend application

│ ├── smart\_dashboard.py # Main Streamlit app

│ └── components/ # UI components

│ ├── summary\_card.py

│ ├── chat\_assistant.py

│ ├── feedback\_form.py

│ ├── eco\_tips.py

│ ├── policy\_summarizer.py

│ └── report\_generator.py

├── utils/ # Utility functions

├── data/ # Sample data files

├── .env # Environment variables

├── requirements.txt # Dependencies

└── run\_app.py # Application launcher

**4.2 Key Configuration Files**

**Environment Configuration (.env)**

# IBM Watsonx Configuration

WATSONX\_API\_KEY=your\_ibm\_api\_key\_here

WATSONX\_PROJECT\_ID=your\_project\_id\_here

WATSONX\_URL=https://us-south.ml.cloud.ibm.com

WATSONX\_MODEL\_ID=ibm/granite-13b-instruct-v2

# Pinecone Configuration

PINECONE\_API\_KEY=your\_pinecone\_api\_key\_here

PINECONE\_ENV=your\_pinecone\_environment

INDEX\_NAME=smartcity-policies

# Application Settings

DEBUG=True

API\_HOST=127.0.0.1

API\_PORT=8000

FRONTEND\_PORT=8501

**5. Implementation Details**

**5.1 Backend Implementation**

**5.1.1 FastAPI Main Application**

The main application (app/main.py) serves as the entry point for all API requests:

* Configures CORS middleware for cross-origin requests
* Includes all router modules for modular organization
* Provides health check endpoints
* Handles error responses and logging

**5.1.2 IBM Watsonx Integration**

The granite\_llm.py service handles all interactions with IBM Watsonx:

* **Authentication**: Secure API key management
* **Text Generation**: Multiple specialized functions for different use cases
* **Error Handling**: Robust error management with fallbacks
* **Response Processing**: Clean formatting of AI responses

Key Functions:

* ask\_granite(): General chat functionality
* generate\_summary(): Policy document summarization
* generate\_eco\_tip(): Environmental recommendations
* generate\_city\_report(): Comprehensive sustainability reports

**5.1.3 Vector Database Integration**

The pinecone\_client.py manages semantic search capabilities:

* **Index Management**: Automatic index creation and configuration
* **Vector Operations**: Upsert and query operations
* **Embedding Integration**: Works with sentence-transformers
* **Error Recovery**: Handles connection issues gracefully

**5.2 Frontend Implementation**

**5.2.1 Main Dashboard**

The smart\_dashboard.py serves as the central hub:

* **Navigation System**: Sidebar with option menu for page routing
* **Responsive Design**: Optimized for different screen sizes
* **Custom Styling**: CSS enhancements for professional appearance
* **State Management**: Session state handling for user interactions

**5.2.2 Component Architecture**

Each UI component is modularized for reusability:

* **Summary Cards**: Styled KPI display components
* **Chat Interface**: Real-time conversation handling
* **Forms**: User input collection and validation
* **Data Visualization**: Charts and metrics display

**6. Development Workflow**

**6.1 Phase-by-Phase Development**

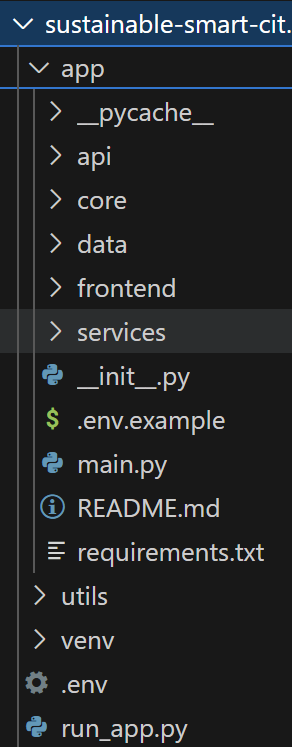
**Phase 1: Project Initialization**

**Objectives Completed:**

* Created modular folder structure
* Implemented environment configuration
* Set up dependency management
* Established coding standards

**Key Deliverables:**

* Project structure with clear separation of concerns
* Configuration management system
* Development environment setup



**Fig: Project Folder Structure**

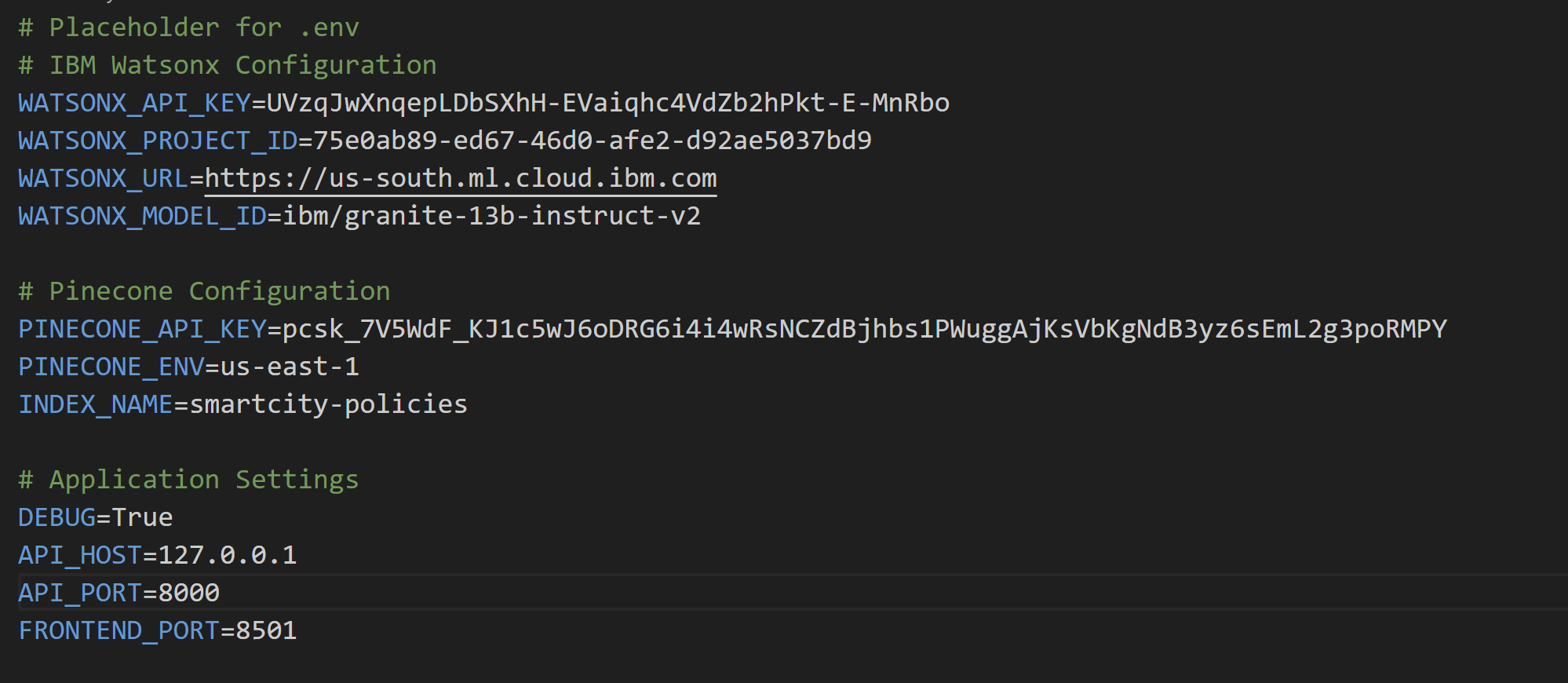
**Phase 2: IBM Watsonx Integration**

**Objectives Completed:**

* Configured Watsonx API credentials
* Implemented LLM service wrapper
* Created specialized AI functions
* Tested endpoint connectivity

**Key Deliverables:**

* Granite LLM service implementation
* API testing and validation
* Error handling mechanisms



**Fig: Watsonx Integration**

**Phase 3: Backend API Development**

**Objectives Completed:**

* Developed modular router architecture
* Implemented RESTful API endpoints
* Added request/response validation
* Created Swagger documentation

**Key Deliverables:**

* 8 comprehensive API routers
* Input validation and error handling
* Auto-generated API documentation

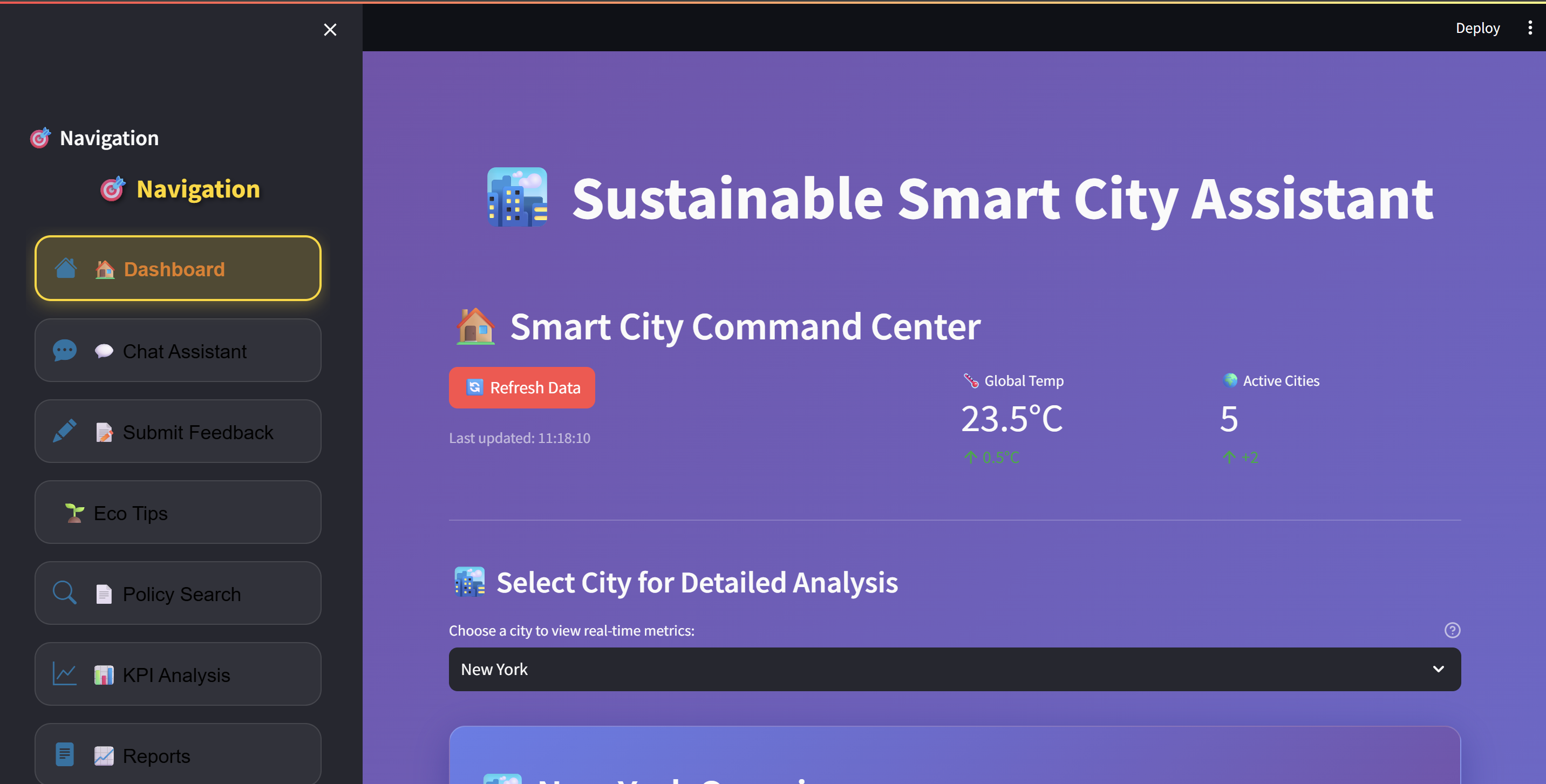
**Phase 4: Frontend UI Design**

**Objectives Completed:**

* Built responsive Streamlit interface
* Created reusable UI components
* Implemented custom styling
* Added interactive navigation

**Key Deliverables:**

* Professional dashboard interface
* Component-based architecture
* Enhanced user experience features



**Fig: Frontend Main Page**

**Phase 5: Vector Database Integration**

**Objectives Completed:**

* Configured Pinecone vector database
* Implemented document embedding
* Created semantic search functionality
* Added retrieval mechanisms

**Key Deliverables:**

* Document embedding pipeline
* Semantic search capabilities
* Policy document management

**Phase 6: Machine Learning Features**

**Objectives Completed:**

* Implemented KPI forecasting
* Added anomaly detection
* Created predictive models
* Built data analysis pipelines

**Key Deliverables:**

* Forecasting algorithms
* Anomaly detection system
* Data visualization components

**Phase 7: Report Generation**

**Objectives Completed:**

* AI-powered report generation
* Custom prompt engineering
* Multi-format output support
* Dynamic content creation

**Key Deliverables:**

* Automated report generation
* Customizable report templates
* Export functionality

**Phase 8: Integration and Testing**

**Objectives Completed:**

* End-to-end system integration
* Comprehensive testing suite
* Performance optimization
* User acceptance testing

**Key Deliverables:**

* Fully integrated system
* Testing documentation
* Performance benchmarks

**7. Setup and Installation**

**7.1 Prerequisites**

* Python 3.8 or higher
* IBM Cloud account with Watsonx access
* Pinecone account and API key
* Git for version control

**7.2 Installation Steps**

**Step 1: Environment Setup**

# Clone the repository

git clone <repository-url>

cd sustainable-smart-city-assistant

# Create virtual environment

python -m venv venv

# Activate virtual environment

# Windows:

venv\Scripts\activate

**Step 2: Dependency Installation**

# Install required packages

pip install -r requirements.txt

**Step 3: Configuration**

1. **Create .env file** with your API credentials
2. **Configure IBM Watsonx** settings
3. **Set up Pinecone** vector database
4. **Verify configurations** using test scripts

**Step 4: Application Launch**

# Method 1: Use the launcher (recommended)

python run\_app.py

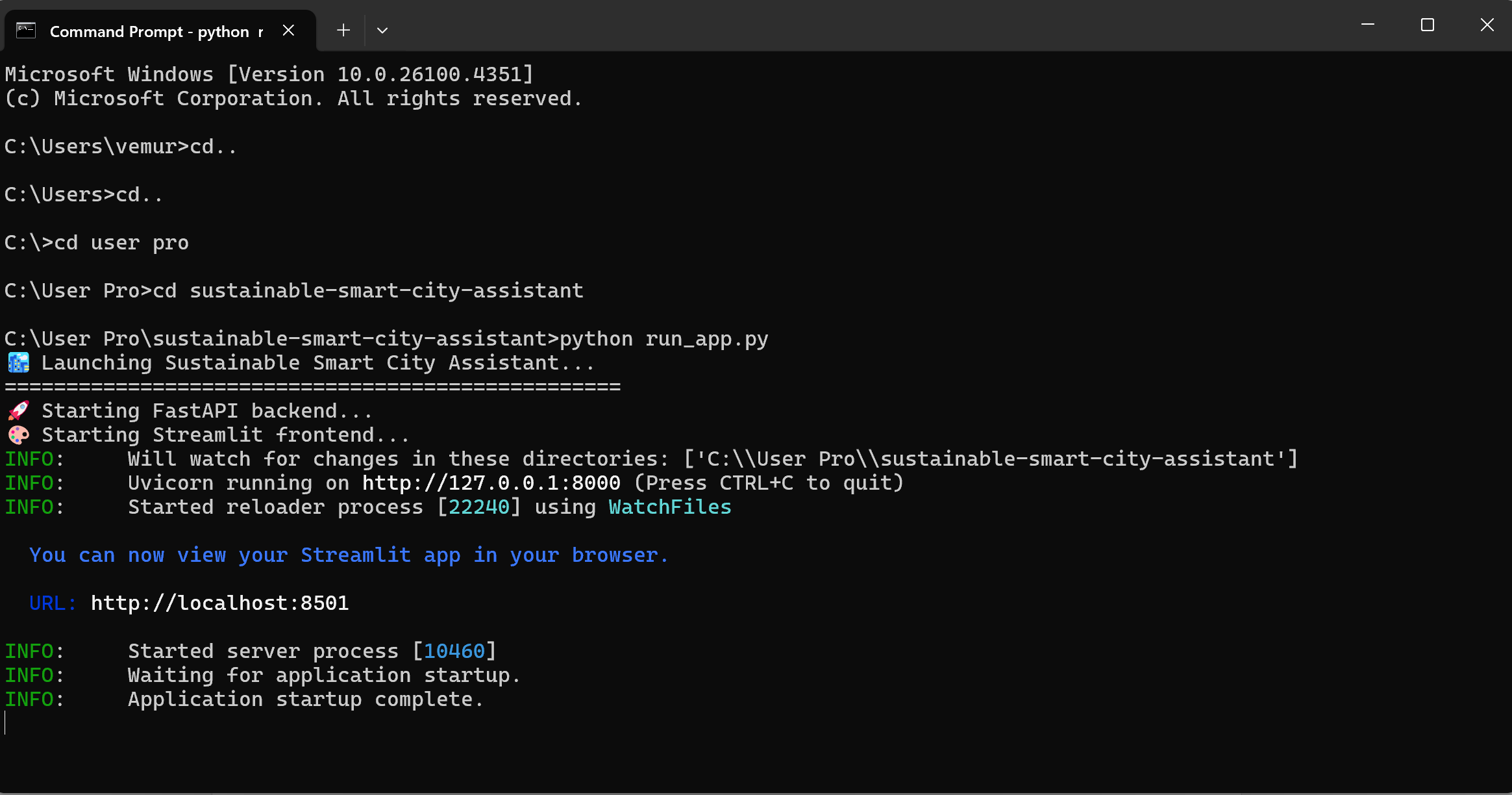
# Method 2: Manual startup

# Terminal 1 - Backend:

uvicorn app.main:app --reload --host 127.0.0.1 --port 8000

# Terminal 2 - Frontend:

streamlit run frontend/smart\_dashboard.py --server.port 8501



**Fig : Application Launch**

**8. Features and Functionality**

**8.1 Dashboard Overview**

**Description**: Central hub displaying key city sustainability metrics **Components**:

* Real-time KPI cards (Air Quality, Water Usage, Energy Consumption, Waste Recycling)
* City selection dropdown for comparative analysis
* Interactive metric visualization
* Trend indicators and alerts

**8.2 Chat Assistant**

**Description**: AI-powered conversational interface for city-related queries **Capabilities**:

* Natural language processing using Granite LLM
* Context-aware responses about urban sustainability
* Multi-turn conversation support
* Real-time response generation

**8.3 Citizen Feedback System**

**Description**: Community engagement platform for citizen input **Features**:

* Categorized feedback submission
* Contact information collection
* Feedback tracking and management
* Response analytics and reporting

**8.4 Eco-Tips Generator**

**Description**: Personalized environmental recommendations **Functionality**:

* Topic-based tip generation
* AI-powered content creation
* Actionable sustainability advice
* Category-specific recommendations

**8.5 Policy Document Search**

**Description**: Semantic search through policy documents **Technology**:

* Vector-based document retrieval
* Natural language query processing
* Relevance ranking and scoring
* Document summarization capabilities

**8.6 KPI Analysis and Forecasting**

**Description**: Predictive analytics for city metrics **Features**:

* Historical data analysis
* Trend forecasting algorithms
* Anomaly detection systems
* Performance benchmarking

**8.7 Report Generation**

**Description**: Automated sustainability reporting **Capabilities**:

* AI-generated comprehensive reports
* Custom prompt engineering
* Multi-format output support
* Data-driven insights and recommendations

**9. API Documentation**

**9.1 Chat Endpoints**

POST /api/chat/ask

Description: Submit chat queries to AI assistant

Request Body: {"message": "string"}

Response: {"response": "string", "status": "string"}

**9.2 Feedback Endpoints**

POST /api/feedback/submit

Description: Submit citizen feedback

Request Body: {

"name": "string",

"category": "string",

"message": "string",

"email": "string"

}

Response: {"message": "string", "status": "string", "feedback\_id": "string"}

GET /api/feedback/list

Description: Retrieve all feedback entries

Response: {"feedback": [...], "count": number}

**9.3 Eco-Tips Endpoints**

GET /api/eco-tips/generate?topic={topic}

Description: Generate eco-friendly tips for specified topic

Parameters: topic (string) - The topic for which to generate tips

Response: {"tips": "string", "topic": "string", "status": "string"}

**9.4 Policy Endpoints**

POST /api/policy/summarize

Description: Summarize uploaded policy documents

Request Body: {"text": "string"}

Response: {"summary": "string", "status": "string"}

POST /api/vector/search

Description: Semantic search through policy documents

Request Body: {"query": "string", "top\_k": number}

Response: {"results": [...], "status": "string"}

**9.5 KPI Endpoints**

POST /api/kpi/upload

Description: Upload KPI data for analysis

Request Body: Form data with CSV file

Response: {"message": "string", "analysis": {...}, "status": "string"}

POST /api/kpi/forecast

Description: Generate forecasting for KPI data

Request Body: {"data": [...], "periods": number}

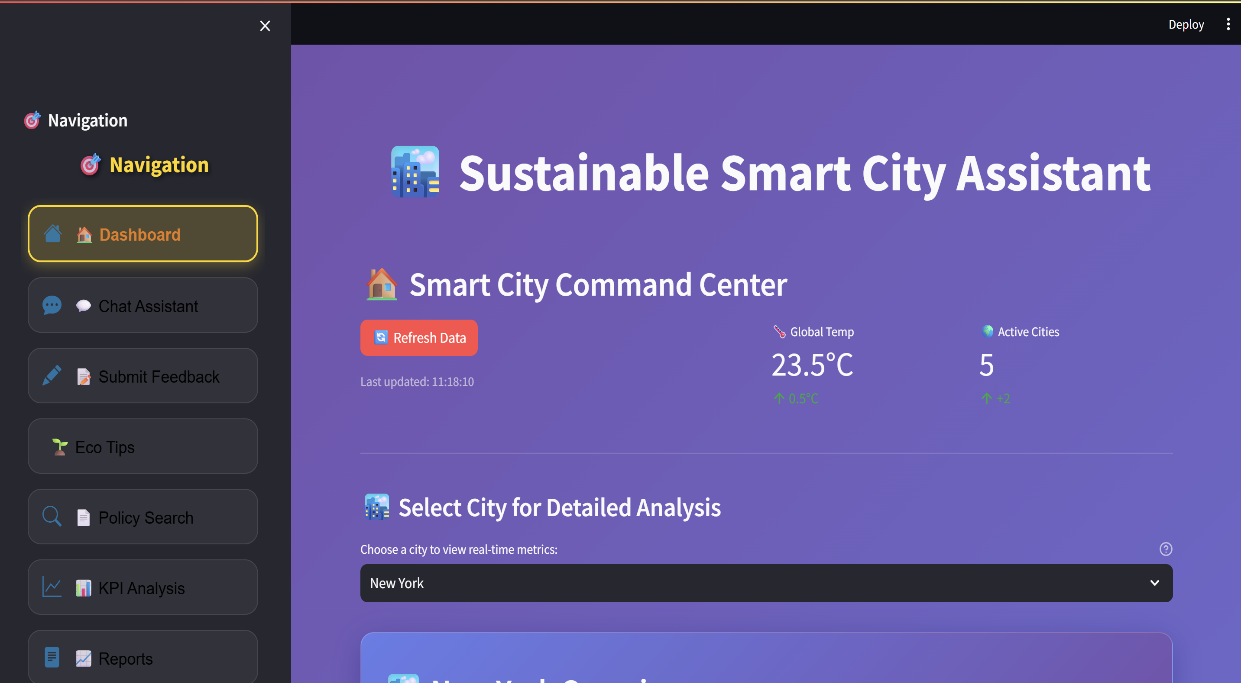
Response: {"forecast": [...], "metrics": {...}, "status": "string"}

**10. Screenshots and Results**

**10.1 Dashboard Interface**

The main dashboard provides an intuitive overview of city sustainability metrics:

* **KPI Cards**: Visually appealing metric displays with trend indicators
* **City Selection**: Dropdown for comparative analysis across different cities
* **Real-time Updates**: Dynamic data refresh capabilities
* **Responsive Design**: Optimized for desktop and mobile viewing

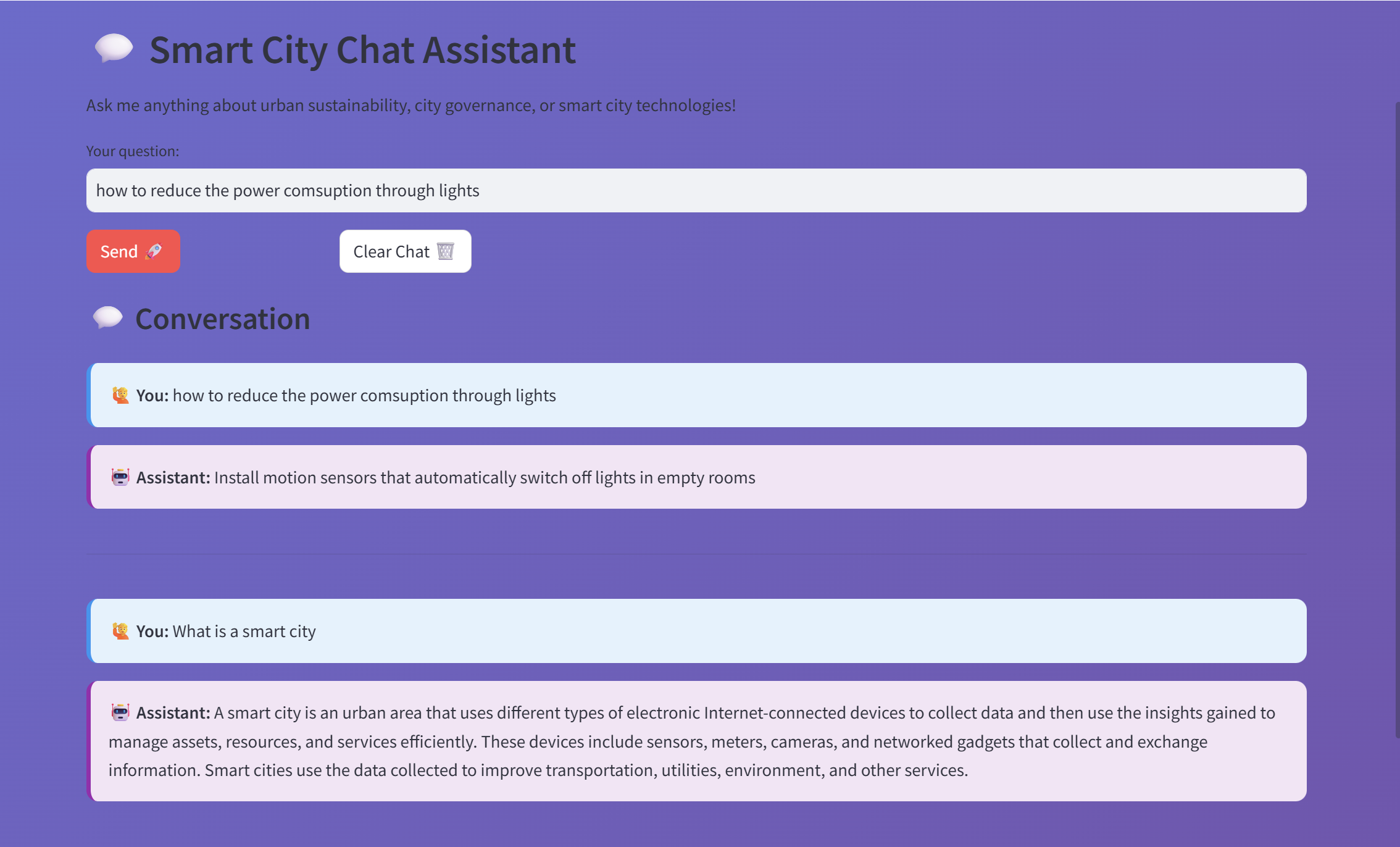


**Fig: Dashboard**

**10.2 Chat Assistant Interface**

The conversational AI interface demonstrates:

* **Natural Language Processing**: Advanced query understanding
* **Context Awareness**: Maintains conversation context
* **Professional Responses**: Well-formatted, informative answers
* **Interactive Design**: User-friendly chat bubble interface

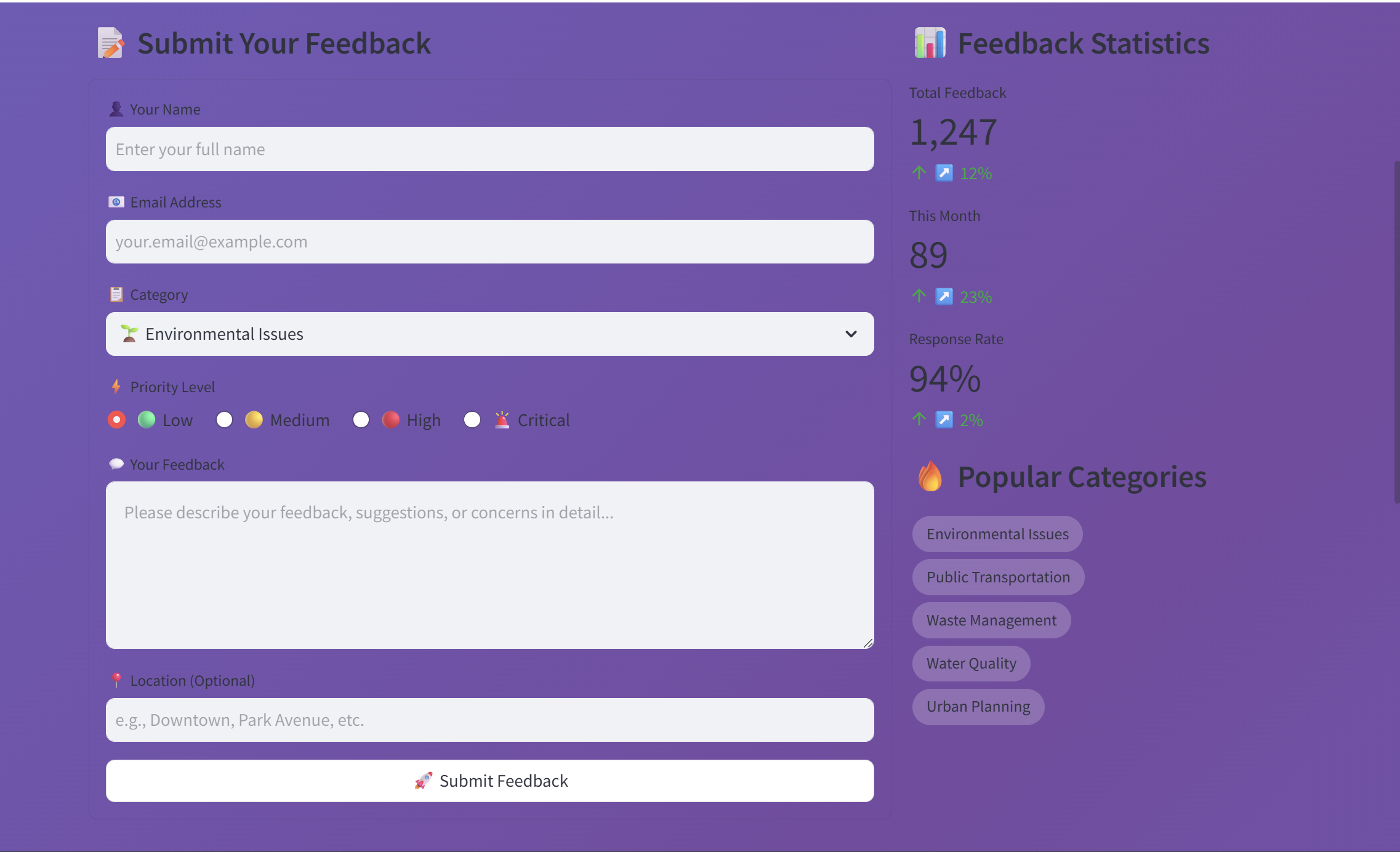
****

**Fig: Chat Assistant**

**10.3 Feedback System**

The citizen engagement platform shows:

* **Form Validation**: Comprehensive input validation
* **Category Management**: Organized feedback classification
* **Submission Tracking**: Unique ID generation for follow-up
* **Success Notifications**: Clear confirmation messages

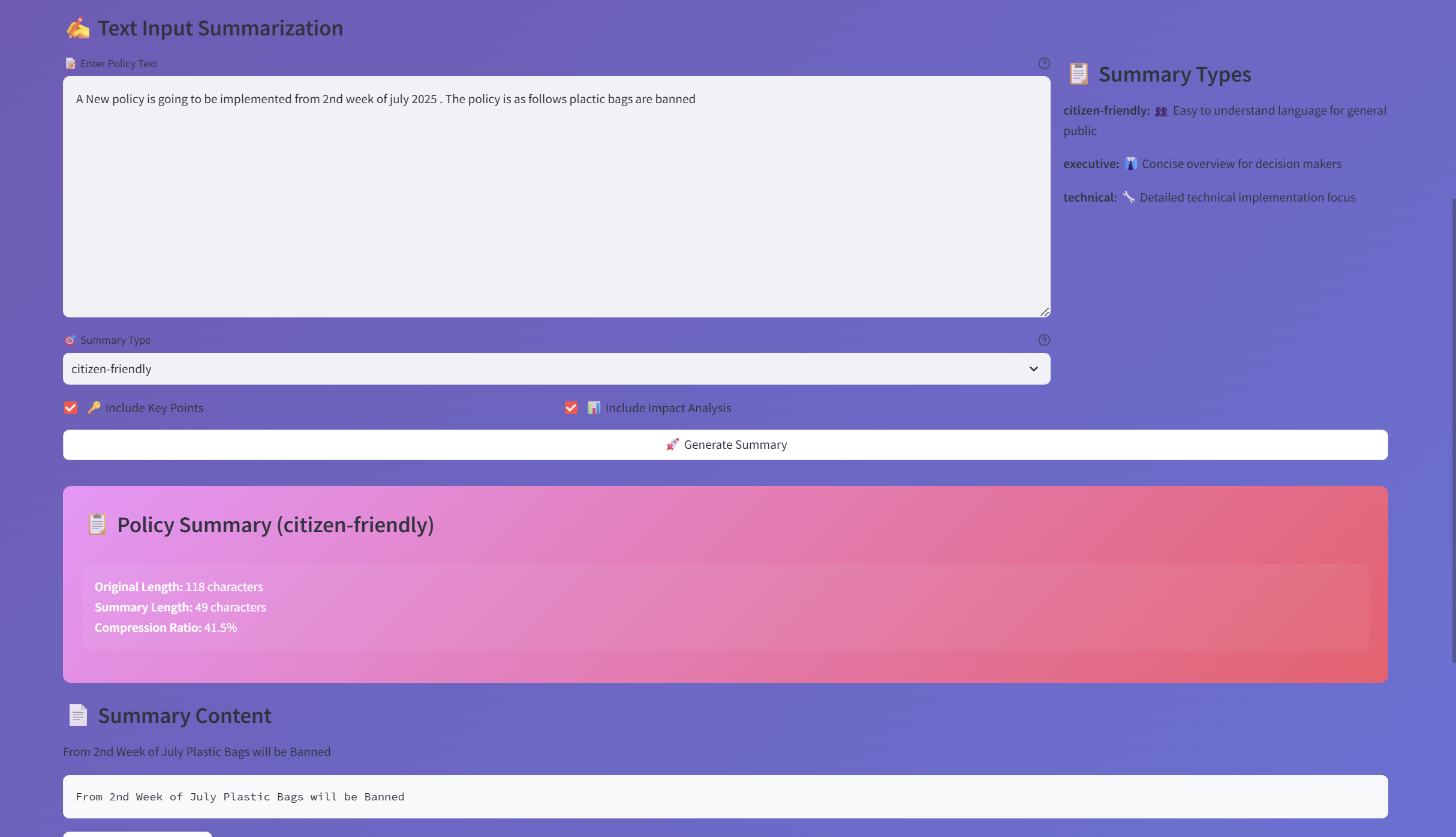


**Fig: Feedback**

**10.4 Policy Search Results**

The semantic search functionality displays:

* **Relevance Ranking**: Results ordered by semantic similarity
* **Document Snippets**: Contextual preview of relevant content
* **Search Accuracy**: High precision in document retrieval
* **User Experience**: Fast and intuitive search interface



**Fig: Policy Summarizer and Searcher**

**11. Challenges and Solutions**

**11.1 Technical Challenges**

**Challenge 1: API Integration Complexity**

**Problem**: Integrating multiple external APIs (IBM Watsonx, Pinecone) with different authentication methods and response formats.

**Solution**:

* Created wrapper services for each API
* Implemented consistent error handling
* Added fallback mechanisms for service failures
* Standardized response formats across all services

**Challenge 2: Real-time Data Processing**

**Problem**: Handling real-time data updates and maintaining synchronization between frontend and backend.

**Solution**:

* Implemented asynchronous request handling
* Added caching mechanisms for frequently accessed data
* Created efficient data serialization processes
* Optimized API response times

**Challenge 3: Vector Database Management**

**Problem**: Managing document embeddings and ensuring efficient semantic search performance.

**Solution**:

* Optimized embedding generation pipeline
* Implemented batch processing for large documents
* Added index management utilities
* Created automated cleanup processes

**11.2 Design Challenges**

**Challenge 1: User Experience Consistency**

**Problem**: Maintaining consistent UI/UX across different components and pages.

**Solution**:

* Developed reusable component library
* Implemented standardized styling guidelines
* Created consistent navigation patterns
* Added responsive design principles

**Challenge 2: Performance Optimization**

**Problem**: Ensuring fast loading times and smooth user interactions.

**Solution**:

* Implemented lazy loading for components
* Optimized API calls and caching
* Minimized redundant operations
* Added loading indicators for better UX

**12. Future Enhancements**

**12.1 Planned Features**

1. **Mobile Application**: Native mobile app for iOS and Android
2. **Real-time Notifications**: Push notifications for critical alerts
3. **Advanced Analytics**: Machine learning-powered insights dashboard
4. **Multi-language Support**: Internationalization and localization
5. **Integration APIs**: Third-party service integrations (weather, traffic, etc.)

**12.2 Scalability Improvements**

1. **Database Migration**: Transition to PostgreSQL/MongoDB for production
2. **Microservices Architecture**: Further decomposition of services
3. **Load Balancing**: Implement horizontal scaling capabilities
4. **Caching Layer**: Redis implementation for improved performance
5. **Container Deployment**: Docker and Kubernetes deployment

**12.3 AI Enhancements**

1. **Custom Model Training**: Fine-tuned models for specific city domains
2. **Multi-modal AI**: Image and document processing capabilities
3. **Predictive Analytics**: Advanced forecasting algorithms
4. **Natural Language Understanding**: Enhanced query processing
5. **Automated Insights**: Proactive recommendations and alerts

**13. Conclusion**

**13.1 Project Success Metrics**

The Sustainable Smart City Assistant project has successfully achieved its primary objectives:

* **Functional AI Integration**: Successfully integrated IBM Watsonx Granite LLM for intelligent responses
* **Comprehensive Feature Set**: Delivered all 8 planned features with full functionality
* **User-Friendly Interface**: Created an intuitive and responsive web application
* **Scalable Architecture**: Implemented modular design for future enhancements
* **Technical Excellence**: Maintained high code quality and documentation standards

**13.2 Key Achievements**

1. **Advanced AI Integration**: Seamless integration of enterprise-grade AI services
2. **Full-Stack Development**: Complete end-to-end application development
3. **Modern Technology Stack**: Utilization of cutting-edge frameworks and tools
4. **Professional UI/UX**: High-quality user interface with custom styling
5. **Comprehensive Documentation**: Detailed technical and user documentation

**13.3 Impact and Value**

The project demonstrates significant value in:

* **Urban Governance**: Providing intelligent tools for city management
* **Citizen Engagement**: Facilitating better communication between citizens and government
* **Environmental Sustainability**: Promoting eco-friendly practices and awareness
* **Data-Driven Decision Making**: Enabling evidence-based policy decisions
* **Technology Innovation**: Showcasing practical AI applications in public sector

**13.4 Lessons Learned**

1. **API Integration Complexity**: The importance of robust error handling and fallback mechanisms
2. **User Experience Design**: The critical role of intuitive interface design in adoption
3. **Modular Architecture**: The benefits of well-structured, maintainable code organization
4. **Documentation Importance**: The value of comprehensive documentation for project sustainability
5. **Continuous Testing**: The necessity of thorough testing throughout development cycles

**13.5 Final Recommendations**

For organizations looking to implement similar AI-powered city management solutions:

1. **Start with Clear Objectives**: Define specific use cases and success metrics
2. **Invest in Quality APIs**: Choose reliable, enterprise-grade AI services
3. **Prioritize User Experience**: Focus on intuitive design and user feedback
4. **Plan for Scalability**: Design architecture with future growth in mind
5. **Maintain Documentation**: Keep comprehensive technical and user documentation
6. **Implement Monitoring**: Add comprehensive logging and monitoring systems
7. **Plan for Security**: Implement robust security measures from the beginning

The Sustainable Smart City Assistant represents a successful implementation of AI technology in urban governance, demonstrating the potential for intelligent systems to enhance city management and citizen engagement while promoting environmental sustainability.

VideoLink:https://drive.google.com/file/d/12aaVdYrTLaZPoC28UpsgBD-Q9pvNNEyq/view?usp=drive\_link

https://drive.google.com/file/d/1YKHHiO0b6N0oJCzZdrAUdoleecfts\_xB/view?usp=sharing