Sustainable Smart City Assistant

1. Introduction

1.1 Project Title

Sustainable Smart City Assistant

1.2 Purpose

This document outlines the functional and non-functional requirements for the "Sustainable Smart City Assistant," an AI-based web platform designed to assist citizens and city administrators in fostering sustainable urban development. It aims to provide tools for information retrieval, forecasting, anomaly detection, feedback collection, and report generation, leveraging advanced AI and data technologies.

1.3 Team Members

- **Team Leader :** Thommandru Ratna Kumari
- **Team member:** Sheik Mubashir Hussen
- **Team member :** Pavan Teja Batthula
- **Team member :** Shaik Mahmmed Fareed
- **Team ID**: LTVIP2025TMID31946
- **Assigned by**: SMART INTERNZ(#SmartBridge)

2. Project Overview

2.1 Purpose

The "Sustainable Smart City Assistant" aims to modernize and streamline city management and citizen engagement by providing a comprehensive, responsive web platform. It will utilize AI to deliver smart responses, generate sustainability tips, enable semantic policy search, forecast key performance indicators (KPIs), detect data anomalies, facilitate citizen feedback, and automate report generation.

2.2 Advanced Features

- **AI-Powered Chat Assistant**: Real-time intelligent responses to user queries using IBM Watsonx Granite LLM.
- **Personalized Eco Tips**: AI-generated sustainability tips based on user-specified keywords.
- **Semantic Policy Search**: Advanced search capabilities within policy documents using Pinecone vector database and sentence embeddings.
- **KPI Forecasting**: Prediction of future KPI values using machine learning models.

- **Data Anomaly Detection**: Identification of outliers in city data.
- **Streamlined Citizen Feedback**: Easy submission and categorization of city issues.
- **Automated Report Generation**: AI-summarized sustainability reports based on KPIs.
- **Interactive Dashboard**: Visual display of key city metrics and multi-city data viewing.

3. Architecture

3.1 Frontend

- **Framework**: Streamlit
- **UI Components**: Streamlit's built-in widgets, styled for rounded cards, icons, and gradient backgrounds. streamlit-option-menu for navigation.
- **API Calls**: Handled by Streamlit's internal mechanisms to interact with the FastAPI backend.

3.2 Backend

- **Framework**: FastAPI
- **Data Validation**: Pydantic for request and response model validation.
- **Environment Management**: python-dotenv for loading API credentials.
- **API Communication**: requests library for external API calls (e.g., to IBM Watsonx).
- **Modular Routers**: Separate routers for each distinct module (Chat, Eco Tips, Policy Search, etc.).

3.3 AI/ML Components

- **LLM Integration**: IBM Watsonx Granite LLM for text generation (Chat Assistant, Eco Tips, Report Generation).
- **Vector Database**: Pinecone for semantic search (Policy Search).
- **Embeddings**: sentence-transformers (MiniLM) for generating vector embeddings of policy documents.
- **Machine Learning**: scikit-learn for KPI forecasting (Linear Regression) and anomaly detection.
- **Data Handling**: pandas for CSV data manipulation.

3.4 Database (Conceptual - for Citizen Feedback and potential KPI storage)

- For Citizen Feedback, a simple in-memory storage or a placeholder for future database integration will be used in the initial version, as per the prompt's scope. The prompt does not explicitly request a database for KPI storage or other modules; data is primarily handled via CSV uploads. If persistence were required,

Firestore would be recommended as per general guidelines.

4. Setup Instructions

4.1 Prerequisites

- Python 3.8+
- Pip (Python package installer)
- Git (for cloning the repository)
- Text Editor/IDE (e.g., VS Code)

4.2 Installation & Running

1. **Clone the repository**: git clone <repository-url>
2. **Navigate to the project directory**: cd sustainable_smart_city_assistant
3. **Create a virtual environment (recommended)**: python -m venv venv
4. **Activate the virtual environment**:
   - Windows: .\venv\Scripts\activate
   - macOS/Linux: source venv/bin/activate
5. **Install dependencies**: pip install -r requirements.txt
6. **Setup .env file**: Create a .env file in the root directory with the provided API credentials.
   - WATSONX_API_KEY
   - WATSONX_PROJECT_ID
   - WATSONX_BASE_URL
   - WATSONX_MODEL_ID
   - PINECONE_API_KEY
   - PINECONE_ENV
   - INDEX_NAME
7. **Run Backend (FastAPI)**:
   - Navigate to the sustainable_smart_city_assistant directory.
   - Run: uvicorn main:app --reload
   - The API will be accessible at http://127.0.0.1:8000 (or similar).
8. **Run Frontend (Streamlit)**:
   - Navigate to the sustainable_smart_city_assistant directory.
   - Run: streamlit run frontend/smart_dashboard.py
   - The Streamlit app will open in your browser (usually http://localhost:8501).

5. Folder Structure

```
sustainable_smart_city_assistant/
├── .env                # Environment variables for API keys
├── main.py             # FastAPI application entry point
```

```
├── config.py           # Configuration loading from .env
├── requirements.txt    # Python dependencies
├── backend/
│   ├── __init__.py     # Makes 'backend' a Python package
│   ├── services/
│   │   ├── __init__.py
│   │   ├── granite_llm.py       # IBM Watsonx LLM integration
│   │   ├── document_embedder.py  # Pinecone and sentence-transformers
│   │   ├── ml_models.py         # ML models for forecasting and anomaly detection
│   ├── routers/
│   │   ├── __init__.py
│   │   ├── chat_router.py         # API endpoints for Chat Assistant
│   │   ├── eco_tips_router.py     # API endpoints for Eco Tips
│   │   ├── policy_router.py       # API endpoints for Policy Search
│   │   ├── kpi_forecasting_router.py # API endpoints for KPI Forecasting
│   │   ├── anomaly_detection_router.py # API endpoints for Anomaly Detection
│   │   ├── feedback_router.py     # API endpoints for Citizen Feedback
│   │   ├── report_generator_router.py # API endpoints for Report Generator
│   ├── models/
│   │   ├── __init__.py
│   │   ├── pydantic_models.py    # Pydantic models for API request/response
├── frontend/
│   ├── smart_dashboard.py     # Streamlit main application
│   ├── components/
│   │   ├── __init__.py
│   │   ├── chat_assistant.py     # Streamlit UI for Chat Assistant
│   │   ├── eco_tips.py          # Streamlit UI for Eco Tips
│   │   ├── policy_search.py      # Streamlit UI for Policy Search
│   │   ├── kpi_forecasting.py    # Streamlit UI for KPI Forecasting
│   │   ├── anomaly_detection.py  # Streamlit UI for Anomaly Detection
│   │   ├── citizen_feedback.py   # Streamlit UI for Citizen Feedback Form
│   │   ├── report_generator_ui.py# Streamlit UI for Report Generator
│   │   ├── dashboard_display.py  # Streamlit UI for Dashboard
```

6. Functional Requirements

6.1 Chat Assistant

- **FR-1.1**: The system shall provide an input field for users to type their queries.

- **FR-1.2**: Upon submission, the system shall send the user's query to the /chat/ask FastAPI endpoint.
- **FR-1.3**: The system shall display the AI-generated response from IBM Watsonx Granite LLM below the input field.
- **FR-1.4**: The input and output boxes shall have rounded edges and padding for improved aesthetics.

6.2 Eco Tips

- **FR-2.1**: The system shall provide an input field for users to enter a keyword.
- **FR-2.2**: Upon submission of a keyword, the system shall call a FastAPI endpoint (e.g., /eco-tips/generate).
- **FR-2.3**: The system shall display 5 AI-generated sustainability tips related to the keyword, generated by IBM Watsonx Granite LLM.

6.3 Policy Search

- **FR-3.1**: The system shall allow users to upload a .txt policy document.
- **FR-3.2**: Upon upload, the backend shall process the document, chunk it, embed the chunks using sentence-transformers, and upsert them into a Pinecone vector database.
- **FR-3.3**: The system shall provide an input field for users to enter a semantic search query.
- **FR-3.4**: Upon query submission, the system shall perform a semantic search against the Pinecone index.
- **FR-3.5**: The system shall display relevant policy document chunks with their similarity scores.

6.4 KPI Forecasting

- **FR-4.1**: The system shall allow users to upload a CSV file containing KPI data.
- **FR-4.2**: The user shall be able to specify the target column for forecasting.
- **FR-4.3**: Upon submission, the backend shall use Linear Regression (or a suitable scikit-learn model) to forecast the next value of the specified KPI.
- **FR-4.4**: The system shall display the forecasted KPI value to the user.

6.5 Anomaly Detection

- **FR-5.1**: The system shall allow users to upload a CSV file containing data for anomaly detection.
- **FR-5.2**: Users shall be able to select specific columns for anomaly detection, or the system shall default to all numeric columns.
- **FR-5.3**: Upon submission, the backend shall apply an anomaly detection algorithm (e.g., Isolation Forest from scikit-learn) to identify outliers.

- **FR-5.4**: The system shall display the identified anomalous data points/rows to the user.

6.6 Citizen Feedback Form

- **FR-6.1**: The system shall provide a form with fields for "Name," "Category," and "Message."
- **FR-6.2**: Users shall be able to submit their feedback through this form.
- **FR-6.3**: Upon submission, the system shall provide a confirmation message. (For this project, feedback will be stored in memory or a simple list on the backend, not persisted to a database, as per the prompt's implied scope.)

6.7 Report Generator

- **FR-7.1**: The system shall provide input fields for users to provide KPI summaries or raw data and additional notes.
- **FR-7.2**: Users shall be able to select a desired report length (e.g., short, medium, long).
- **FR-7.3**: Upon submission, the backend shall use IBM Watsonx Granite LLM to auto-generate a sustainability report based on the provided inputs.
- **FR-7.4**: The system shall display the generated report text.

6.8 Dashboard

- **FR-8.1**: The system shall display key KPI summary cards (e.g., mock data for water usage, energy consumption, air quality).
- **FR-8.2**: The system shall allow users to switch between different "cities" to view their respective KPI data (using mock or predefined data).

7. Non-Functional Requirements

7.1 Performance

- **NFR-7.1.1**: The Chat Assistant response time shall be under 5 seconds for typical queries.
- **NFR-7.1.2**: Policy document embedding and upsert operations should complete within a reasonable time proportional to document size.
- **NFR-7.1.3**: KPI forecasting and anomaly detection for small to medium CSV files (e.g., up to 1MB) should complete within 5-10 seconds.

7.2 Usability

- **NFR-7.2.1**: The user interface shall be intuitive and easy to navigate for all modules.
- **NFR-7.2.2**: The design shall be clean, readable, and consistent across all pages, using rounded cards, icons, and gradient backgrounds where appropriate.

- **NFR-7.2.3**: Error messages shall be clear and user-friendly, guiding the user on how to resolve issues.

7.3 Reliability

- **NFR-7.3.1**: The system shall handle API call failures gracefully, providing informative error messages to the user.
- **NFR-7.3.2**: Data processing (CSV parsing, text chunking) shall be robust to common data inconsistencies.

7.4 Scalability (Conceptual)

- **NFR-7.4.1**: The FastAPI backend should be designed to handle multiple concurrent requests.
- **NFR-7.4.2**: The use of Pinecone for vector search inherently provides scalability for policy document storage and retrieval.

7.5 Security

- **NFR-7.5.1**: API keys and sensitive credentials shall be loaded securely from environment variables (.env file) and not hardcoded.
- **NFR-7.5.2**: Input validation shall be performed on all user inputs to prevent common vulnerabilities.

7.6 Maintainability

- **NFR-7.6.1**: The codebase shall be modular, with clear separation of concerns (frontend, backend services, routers, models).
- **NFR-7.6.2**: Code shall be clean, readable, and extensively commented.
- **NFR-7.6.3**: Dependencies shall be managed via requirements.txt.

8. API Documentation (Backend Endpoints)

- **POST /chat/ask**
  - **Description**: Get a smart response from the AI chat assistant.
  - **Request Body**: ChatRequest (query: str)
  - **Response Body**: ChatResponse (response: str)
- **POST /eco-tips/generate**
  - **Description**: Generate 5 sustainability tips based on a keyword.
  - **Request Body**: EcoTipsRequest (keyword: str)
  - **Response Body**: EcoTipsResponse (tips: List[str])
- **POST /policy/upload**
  - **Description**: Upload a policy document (.txt) for semantic search.
  - **Request Body**: File upload (multipart/form-data)
  - **Response Body**: PolicyUploadResponse (message: str, index_name: str,

vector_count: int)

- **POST /policy/search**
  - **Description**: Perform a semantic search within uploaded policy documents.
  - **Request Body**: PolicySearchRequest (query: str, top_k: Optional[int])
  - **Response Body**: PolicySearchResponse (results: List[PolicySearchResult])
- **POST /kpi/forecast**
  - **Description**: Forecast the next KPI value from CSV data.
  - **Request Body**: KPIForecastRequest (csv_data: str, target_column: str)
  - **Response Body**: KPIForecastResponse (forecasted_value: float, message: str)
- **POST /anomaly/detect**
  - **Description**: Detect anomalies in uploaded CSV data.
  - **Request Body**: AnomalyDetectionRequest (csv_data: str, columns_to_check: Optional[List[str]])
  - **Response Body**: AnomalyDetectionResponse (anomalies: List[AnomalyResult], message: str)
- **POST /feedback/submit**
  - **Description**: Submit citizen feedback.
  - **Request Body**: CitizenFeedbackRequest (name: str, category: str, message: str)
  - **Response Body**: CitizenFeedbackResponse (message: str)
- **POST /report/generate**
  - **Description**: Auto-generate a sustainability report.
  - **Request Body**: ReportGenerateRequest (kpi_summary: str, additional_notes: Optional[str], report_length: Optional[str])
  - **Response Body**: ReportGenerateResponse (report_text: str, message: str)

9. Testing (Planned)

- **Unit Testing**: Individual components and functions (e.g., LLM service, embedding utility, ML models) will be tested in isolation.
- **Integration Testing**: Verify the correct interaction between frontend and backend endpoints, and between backend services and external APIs (Watsonx, Pinecone).
- **Functional Testing**: End-to-end testing of each module (Chat Assistant, Eco Tips, etc.) to ensure they meet the specified requirements.
- **UI/UX Testing**: Manual verification of the Streamlit interface for responsiveness, usability, and adherence to design guidelines (rounded cards, gradients).

10. Screenshots or Demo (Placeholder)

- Demo video: Recording 2025-06-29 173346 (1).mp4
- Code repository: github repo link

11. Conclusion

- The "Sustainable Smart City Assistant" project is designed as a modular, AI-driven platform to empower both citizens and administrators in promoting sustainable urban living. By leveraging a robust FastAPI backend and an intuitive Streamlit frontend, coupled with advanced AI capabilities from IBM Watsonx Granite LLM and Pinecone, the assistant will provide essential tools for informed decision-making, efficient resource management, and active community engagement. This SRS lays the foundation for developing a system that is not only functionally rich but also user-friendly, reliable, and scalable, with clear pathways for future enhancements to further enrich its impact on smart city initiatives.