DOCUMENTATION OF THE PROJECT SMART CITY AI ASSISTANT

INTRODUCTION:

ProjectTitle: SMART CITY AI ASSISTANT USING IBM GRANITE LLM

Team Leader: Gaytri P

Team Member: IsrathSulthana J

Team Member: Pavithara D

Team Member: Keerthika M

Team Member: Sam S

PROJECT DESCRIPTION / PURPOSE:

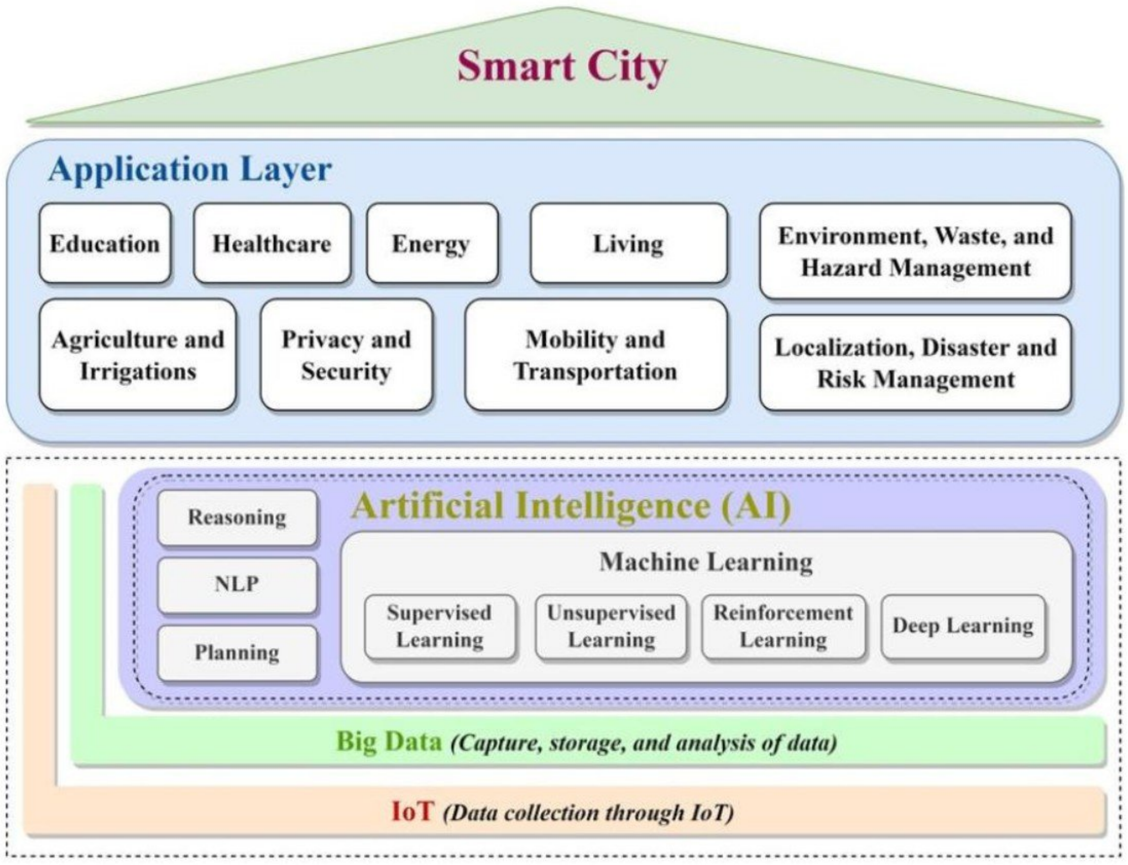
The purpose ofa Smart City Assistant is to empower cities and their residents to thrive in a more eco-conscious and connected urban environment. By leveraging AI and real-time data, the assistant helps optimize essential resources like energy, water, and waste, while also guiding sustainable behaviors among citizens through personalized tips and services. For city officials, it serves as a decision-making partner—offering clear insights, forecasting tools, and summarizations of complex policies to support strategic planning. Ultimately, this assistant bridges technology, governance, and community engagement to foster greener cities that are more efficient, inclusive, and resilient.

FEATURES:

- Conversational Interface: Natural language interaction – Allows citizens and officials to ask questions, get updates, and receive guidance in plain language.
- Policy Summarization: Simplified policy understanding – Converts lengthy government documents into concise, actionable summaries.
- Resource Forecasting: Predictive analytics – Estimates future energy, water, and waste usage using historical and real-time data.
- Eco-Tip Generator: Personalized sustainability advice – Recommends daily actions to reduce environmental impact based on user behavior.
- Citizen Feedback Loop: Community engagement – Collects and analyzes public input to inform city planning and service improvements.
- KPI Forecasting: Strategic planning support – Projects key performance indicators to help officials track progress and plan ahead.

- Anomaly Detection: Early warning system – Identifies unusual patterns in sensor or usage data to flag potential issues.
- Multimodal Input Support: Flexible data handling – Accepts text, PDFs, and CSVs for document analysis and forecasting.
- Streamlit or Gradio UI: User-friendly interface – Provides an intuitive dashboard for both citizens and city officials to interact with the assistant.

ARCHITECTURE:


Smart City
Application Layer
Education
Healthcare
Energy
Living
Environment, Waste, and Hazard Management
Agriculture and Irrigations
Privacy and Security
Mobility and Transportation
Localization, Disaster and Risk Management
Artificial Intelligence (AI)
Reasoning
NLP
Planning
Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Deep Learning
Big Data (Capture, storage, and analysis of data)
IoT (Data collection through IoT)

The architecture consists of three main layers:

• Frontend (Streamlit): Interactive web UI with dashboards, file uploads, chat, feedback, and report viewing. Modular pages for scalability.
• Backend (FastAPI): REST framework handling document processing, chat, eco tips, reports, and vector embeddings. Optimized for async performance with Swagger support.
• External Services: IBM Watsonx Granite models for LLM, Pinecone for vector search, Scikit-learn for forecasting and anomaly detection.

SETUP INSTRUCTIONS:

Prerequisites:
- Python 3.9 or later
- pip and virtual environment tools
- API keys for IBM Watsonx and Pinecone
- Internet access to access cloud services

Installation Process:
- Clone the repository
- Install dependencies from requirements.txt
- Create a .env file and configure credentials
- Run the backend server using FastAPI
- Launch the frontend via Streamlit
- Upload data and interact with the modules

PROJECT MILESTONES:

1. Phase 1 – Initialization: Set up modular folder structure, environment configs, and Pinecone vector index.
2. Phase 2 – Watsonx Integration: Configured API keys, models, and validated endpoints via Swagger.
3. Phase 3 – Backend APIs: Built modular routers (chat, feedback, eco tips, KPIs, vectors) with robust testing.
4. Phase 4 – Frontend UI: Designed Streamlit dashboard with modular components, navigation, and styled UI.
5. Phase 5 – Pinecone & Embeddings: Implemented document embedding and retrieval using sentence-transformers.
6. Phase 6 – Reports & Deployment: Granite LLM-powered report generation, PDF/Markdown support, and integration testing.

FOLDER STRUCTURE & RUNNING THE APPLICATION:

Folder Structure:
- app/ – FastAPI backend logic
- app/api/ – Modular API routes (chat, feedback, report, document vectorization)
- ui/ – Streamlit frontend pages
- smart_dashboard.py – Entry script for dashboard
- granite_llm.py – Handles Watsonx Granite model
- document_embedder.py – Embeds documents in Pinecone
- kpi_file_forecaster.py – Forecasts future trends
- anomaly_file_checker.py – Detects anomalies
- report_generator.py – Generates sustainability reports

Running:
- Launch the FastAPI server
- Run the Streamlit dashboard
- Navigate through sidebar pages
- Upload documents or CSVs, interact with assistant, and view reports and predictions

APIDOCUMENTATION:

Available APIs include:
- POST /chat/ask
- POST /upload-doc
- GET /search-docs
- GET /get-eco-tips
- POST /submit-feedback

(All endpoints documented in Swagger UI).

AUTHENTICATION:

Implemented via Swagger UI for testing. Supports JWT/API key authentication, OAuth2 with IBM Cloud, and role-based access. Planned: user sessions and history tracking.

USER INTERFACE:

Minimalist Streamlit/Gradio UI with sidebar navigation, KPI visualizations, chat/eco tips tabs, PDF report downloads, and real-time updates.

TESTING:

Testing phases included unit testing, API testing, manual file/chat tests, and edge case handling. All functions validated for both offline and online modes.

CODE:

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""
    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {str(e)}"

def eco_tips_generator(problem_keywords):
    prompt = f"Generate practical and actionable eco-friendly tips for sustainable living
related to: {problem_keywords}. Provide specific solutions and suggestions:"
    return generate_response(prompt, max_length=1000)

def policy_summarization(pdf_file, policy_text):
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)
        summary_prompt = f"Summarize the following policy document and extract the most
```

```
importantpoints, key provisions, and implications:\n\n{content}"
  else:
    summary_prompt = f"Summarize the following policy document and extract the most
importantpoints, key provisions, and implications:\n\n{policy_text}"
  returngenerate_response(summary_prompt, max_length=1200)

#CreateGradio interface
withgr.Blocks() as app:
  gr.Markdown("# Eco Assistant & Policy Analyzer")
  withgr.Tabs():
    withgr.TabItem("Eco Tips Generator"):
      withgr.Row():
        withgr.Column():
          keywords_input = gr.Textbox(
            label="Environmental Problem/Keywords",
            placeholder="e.g., plastic, solar, water waste, energy saving...",
            lines=3
          )
          generate_tips_btn = gr.Button("Generate Eco Tips")
        withgr.Column():
          tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)
          generate_tips_btn.click(eco_tips_generator, inputs=keywords_input,
outputs=tips_output)
    withgr.TabItem("Policy Summarization"):
      withgr.Row():
        withgr.Column():
          pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])
          policy_text_input = gr.Textbox(
            label="Or paste policy text here",
            placeholder="Paste policy document text...",
            lines=5
          )
          summarize_btn = gr.Button("Summarize Policy")
        withgr.Column():
          summary_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)
          summarize_btn.click(policy_summarization, inputs=[pdf_upload,
policy_text_input], outputs=summary_output)
```

app.launch(share=True)

FUTURE ENHANCEMENTS:

- Multi-language Support
- Advanced Summarization (short + detailed, named entities)
- Contextual Eco Tips (region, climate, sector-based)
- Report Generation with charts and visuals
- Database/Vector Store Integration (Pinecone, FAISS, Weaviate)
- Watsonx Granite API integration
- External APIs for sustainability datasets
- Interactive Dashboard with charts & comparisons
- Authentication & Role-based access
- Mobile-Friendly UI
- Cloud Deployment (AWS/GCP/Azure)
- CI/CD pipeline with Docker/Kubernetes
- Offline Mode for local inference

CONCLUSION:

The Smart City AI Assistant demonstrates how AI and IBM Granite LLM can enhance sustainable governance through eco-friendly recommendations, real-time citizen engagement, anomaly detection, forecasting, and policy summarization. By bridging citizens, governments, and technology, this project ensures smarter, greener, and more resilient cities.