

Project Title - Sustainable Smart City Assistant Using IBM Granite LLM

Introduction :

Team ID : NM2025TMID06825

Team Size : 4

Team Leader : VIMALRAJ V

Team member : SENTHIL KUMAR D

Team member : SUNDARESHWARAN R

Team member : SURESH K

Project Overview:

The Sustainable Smart City Assistant is an AI-powered platform that leverages IBM Watsonx's Granite LLM and modern data pipelines to support urban sustainability, governance, and citizen engagement. It integrates several modules like City Health Dashboard, Citizen Feedback, Document Summarization, Eco-Advice, Anomaly Detection, KPI forecasting and Chat Assistant through a modular FastAPI backend and a Streamlit- based frontend dashboard.

Use Case Scenarios

Policy Search & Summarization

A municipal planner uploads a complex city policy document to the assistant's interface. In seconds, the assistant summarizes it into a concise, citizen-friendly version using IBM Granite LLM. This empowers planners to quickly interpret key points and make informed urban decisions.

Citizen Feedback Reporting

A resident notices a burst water pipe on a city street. Instead of calling helplines, they submit a report through the assistant's feedback form. The issue is logged instantly with category tagging (e.g., "Water") and can be reviewed by city administrators.

KPI Forecasting

A city administrator uploads last year's water usage KPI CSV. The assistant forecasts next year's consumption using built-in machine learning. This data is used in planning budgets and infrastructure upgrades.

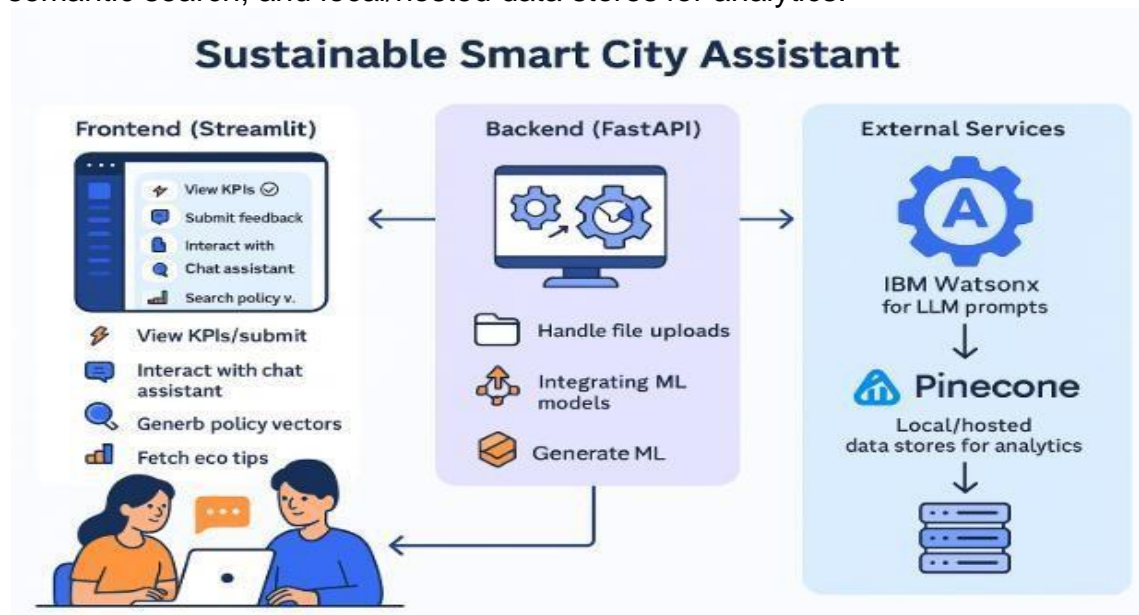
Architecture

The architecture consists of three main layers:

Frontend (Streamlit): Provides a modular dashboard with options to view KPIs, submit feedback, interact with chat assistant, search policy vectors, generate reports, and fetch eco tips.

Backend (FastAPI): Manages API requests, handles file uploads, and integrates with ML models, Pinecone, and Watsonx Granite LLM.

External Services: IBM Watsonx for LLM prompts, Pinecone for vector indexing & semantic search, and local/hosted data stores for analytics.



Project Flow

1. User Input:

Users interact with the **Streamlit frontend dashboard**, where they can:

- Submit **textual prompts** (for chat or policy summaries).
- Upload **policy documents** (.txt, .csv) for summarization and vector search.
- Choose a city to view **real-time KPIs** (water usage, air quality, energy).
- Submit **citizen feedback** (name, category, message).
- Ask sustainability queries via **chat interface**.
- Search for **eco-friendly tips** by entering a topic keyword.

UI Components Involved:

smart_dashboard.py, feedback_form.py, chat_assistant.py, eco_tips.py,
summary_card.py

2. Backend Processing (FastAPI):

Each input request is sent to corresponding **FastAPI endpoints**, where:

- Feedback is stored and categorized through feedback_router.py.
- KPI .csv files are forecasted using internal ML models in kpi_file_forecaster.py.
- Text prompts (from chat, summarizer, eco tips) are sent to **IBM Granite LLM** using the granite_llm.py service.
- Anomaly detection is applied to uploaded datasets using statistical checks.

Key Backend Components:

vector_router.py, chat_router.py, kpi_upload_router.py, granite_llm.py,
pinecone_client.py

3. AI Response Generation:

- The **Watsonx Granite LLM** processes chat queries, summaries, eco tips, and generates human-like natural language responses.
- **ML models** forecast future KPIs or detect anomalies in uploaded files.
- **Pinecone** retrieves the most relevant policy document chunks using semantic search powered by vector similarity.

Output Formats: JSON objects containing text summaries, search results, KPIs, anomaly alerts.

4. Frontend Display:

The **Streamlit frontend** dynamically renders:

- KPI data in **visually enhanced cards** (summary_card.py).
- AI-generated responses (chat, eco tips) directly in user input sections.
- Policy search results in readable formats.
- Submission success or errors through toast messages (e.g., feedback success).

Frontend Enhancements Done: Rounded input cards, Gradient background, Icon-rich sidebar, Themed buttons and layout improvements.

5. User Interaction:

Users are able to:

- Switch cities and compare urban KPIs dynamically.
- Ask follow-up queries in the chat assistant.
- Generate policy summaries and sustainability reports.
- Continuously explore eco tips with varied topics.
- Interact with updated dashboard metrics in real time.

Real-time interaction enabled by:

FastAPI + Streamlit two-way binding with updated backend JSON responses.

Prior Knowledge

You must have the prior knowledge of the following topics to complete this project:

Generative AI Concepts

NLP: https://www.tutorialspoint.com/natural_language_processing/index.htm

Generative AI: https://en.wikipedia.org/wiki/Generative_artificial_intelligence

IBM Watsonx Granite: <https://cloud.ibm.com/watsonx/overview>

LangChain: <https://docs.langchain.com/docs/>

Pinecone: <https://docs.pinecone.io/docs/overview>

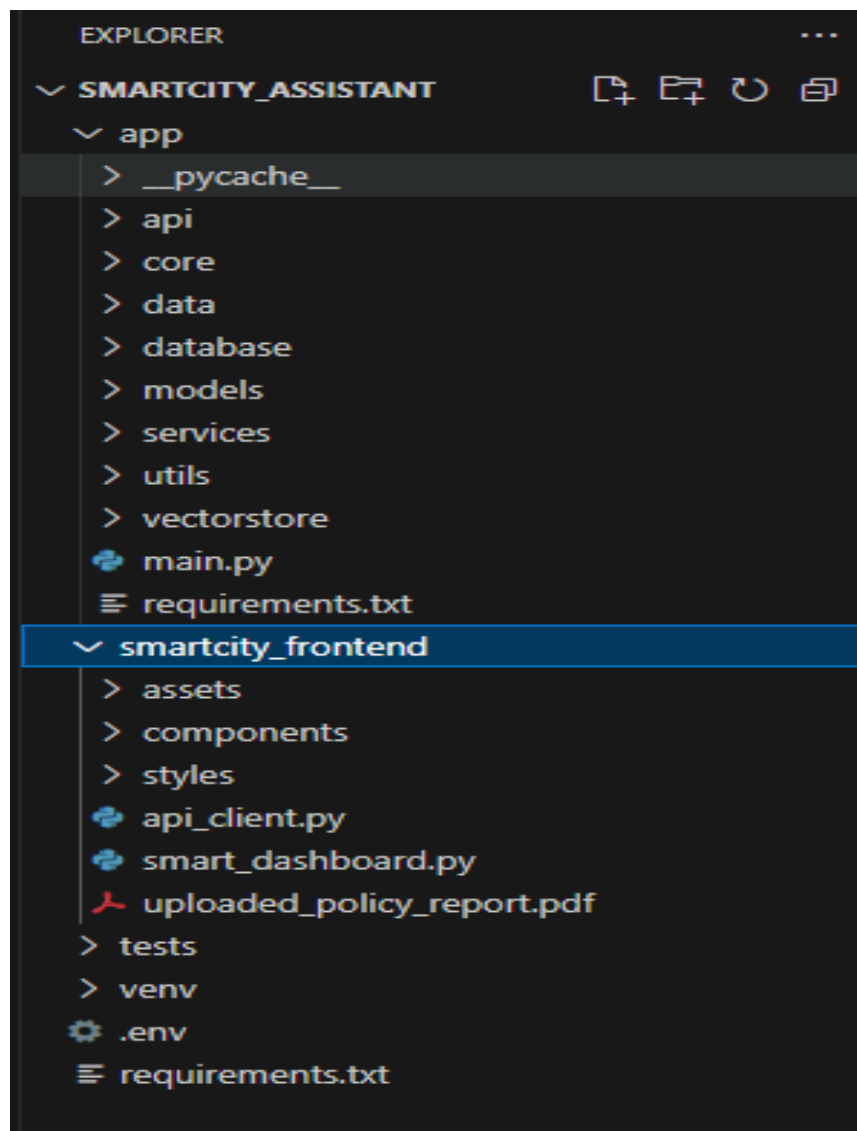
FastAPI: <https://fastapi.tiangolo.com/>

Streamlit: <https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/>

Project Milestones & Development Flow

Phase 1 – Project Initialization

Modular Folder Structure Defined: Created separate folders for app/api, services, vectorstore, core, frontend/components, and utils for organized and scalable development.



Environment Setup:

.env file created with keys for Pinecone and Watsonx.
config.py loads environment variables securely using
pydantic.

.env file

```
.env
1 WATSONX_API_KEY=pAjcxi3Df0g687V0mCe3_Q8TmRKSDlp9wulXo52qwNn5
2 WATSONX_PROJECT_ID=f371addd-61dd-4ff0-882d-571db5a32aea
3 WATSONX_URL=https://eu-de.ml.cloud.ibm.com
4 WATSONX_MODEL_ID=ibm/granite-13b-instruct-v2
5 PINECONE_API_KEY=pcsk_22YGb3_9RY8BMqaUZN55nkxUA7nR7ZyhBnKA1LjW44XtRpkeo43rCmj2yW4HrPhQfQbafu
6 PINECONE_ENV=us-east-1
7 INDEX_NAME=smartcity-policies
```

Config.py file

```
app > core > config.py > Settings > Config
8 class Settings(BaseSettings):
12
13
14     # Watsonx configs
15     WATSONX_API_KEY: str = "pAjcxi3DfOg687V0mCe3_Q8TmRKSDlp9wulXo52qwNn5"
16     WATSONX_PROJECT_ID: str = "f371addd-61dd-4ff0-882d-571db5a32aea"
17     WATSONX_URL: str = "https://eu-de.ml.cloud.ibm.com"
18     WATSONX_MODEL_ID: str = "ibm/granite-13b-instruct-v2"
19
20
21     class Config:
22         env_file = ".env"
23         extra = "allow"
24
25 settings = Settings()
26
```

Pinecone Initialization:

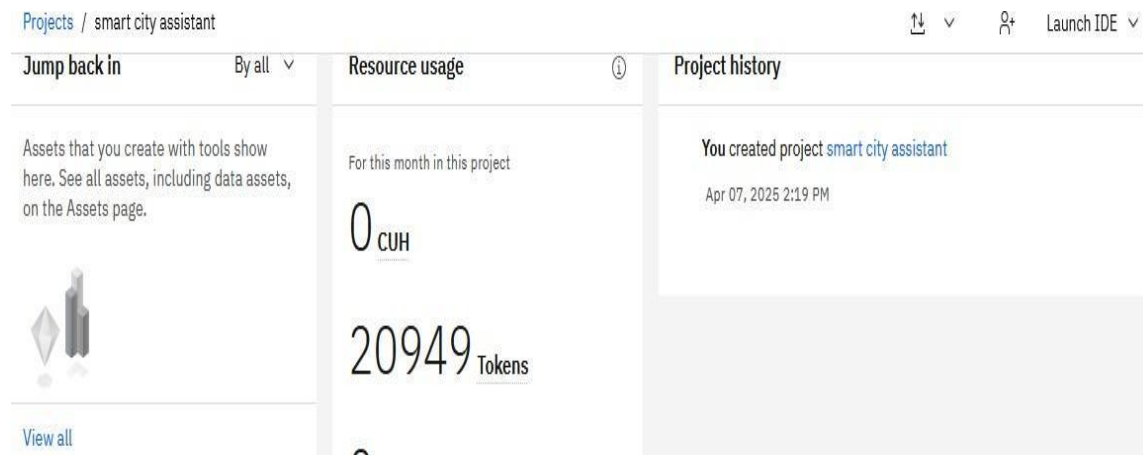
pinecone_client.py written to initialize the Pinecone vector index (smartcity-policies). Ensured creation with correct dimension=384 matching embedding model.

Phase 2 – IBM Watsonx Integration

Watsonx Key & Model

Configuration: Set up .env with:

WATSONX_API_KEY, PROJECT_ID, MODEL_ID



Endpoint Testing:

Validated /chat, /policy/summarize, and /get-eco-tips FastAPI routes using Swagger UI.

FastAPI

0.1.0

OAS 3.1

/openapi.json

Citizen Feedback

POST

/submit-feedback Submit Feedback

Citizen Tips

GET

/get-eco-tips Get Tips

Admin Tools

POST

/generate-report Generate Report

GET

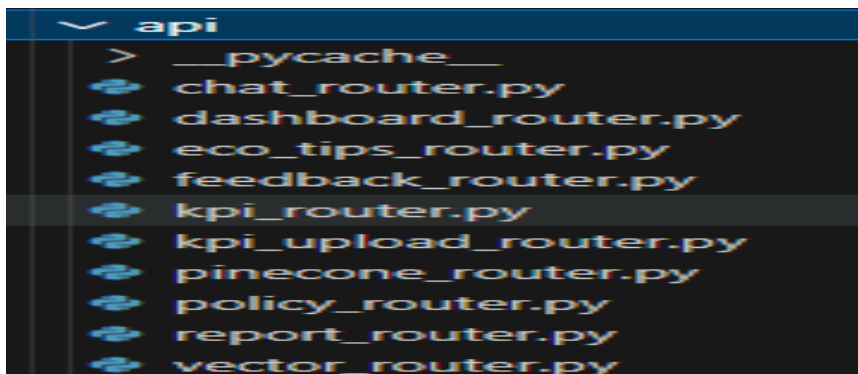
/anomaly-alerts Get Alerts

Phase 3 – Backend API Routers

API Routes Implemented:

Developed modular routers:

- chat_router.py
- feedback_router.py
- eco_tips_router.py
- kpi_upload_router.py
- anomaly_checker.py
- vector_router.py, etc.



Testing & Validation:

Each route tested for:

- JSON payload correctness
- File upload parsing
- Error handling & logging
- Swagger auto-documentation generation

Chat Assistant

POST `/chat/ask-assistant` Ask Question

Policy Summarizer

POST `/policy/summarize-policy` Summarize

GET `/policy/test-llm` Test Llm

GET `/policy/summarize-from-file` Summarize From File

POST `/policy/summarize-uploaded-file` Summarize Uploaded File

POST `/policy/generate-markdown-report` Generate Md From Text

POST `/policy/generate-pdf-report` Generate Pdf From Text

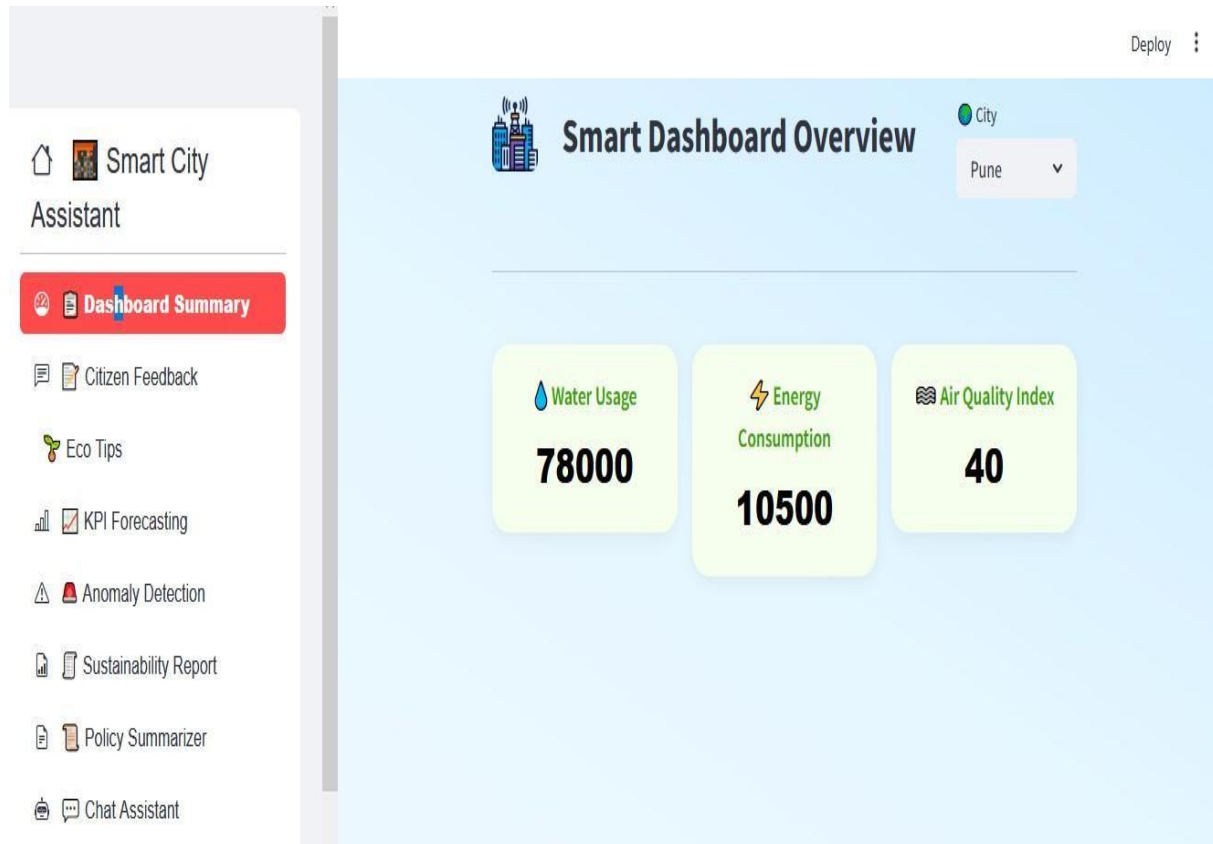
POST `/policy/upload-txt-generate-markdown` Generate Md From Uploaded Txt

POST `/policy/upload-txt-generate-pdf` Generate Pdf From Uploaded Txt

Phase 4 – Frontend UI Design

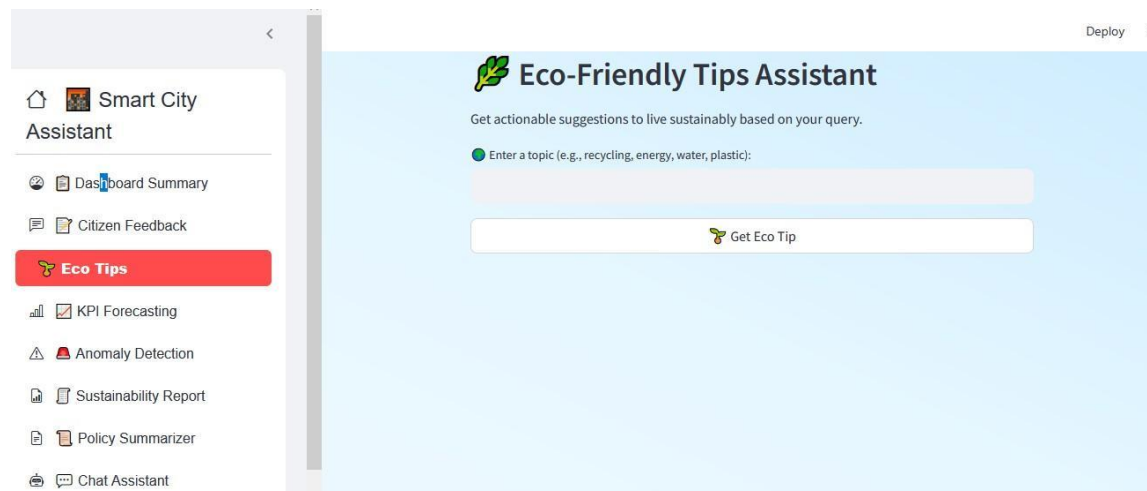
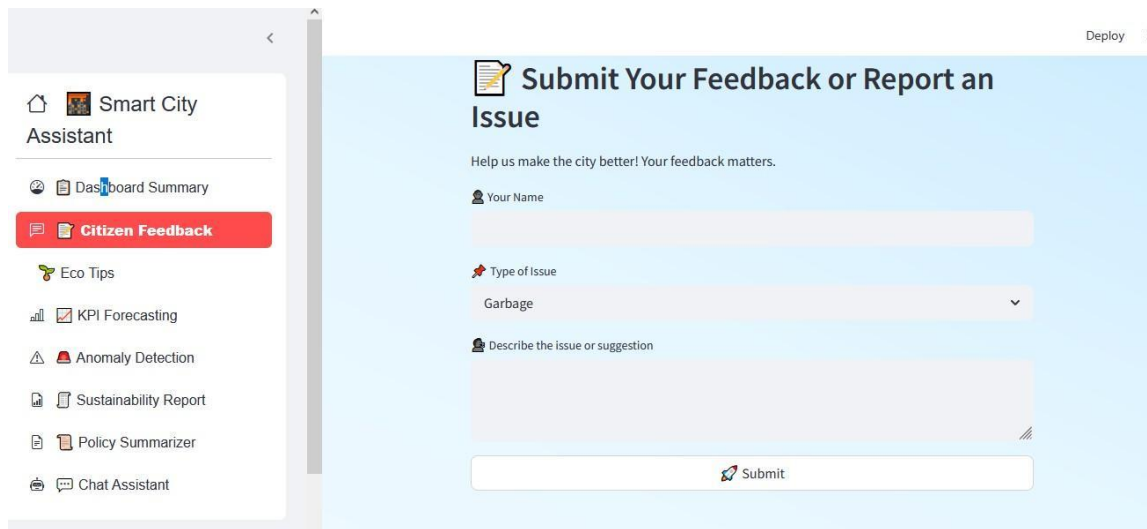
Streamlit UI Structure Implemented:

Created central file `smart_dashboard.py` with conditional rendering for each module using sidebar navigation.



Component Development:

Developed reusable Streamlit components:
`summary_card.py` – Beautiful KPI cards
`chat_assistant.py` – Text prompt and AI reply
`feedback_form.py`, `eco_tips.py`, `report_generator.py`, etc.



UI Enhancements Done:

Gradient backgrounds

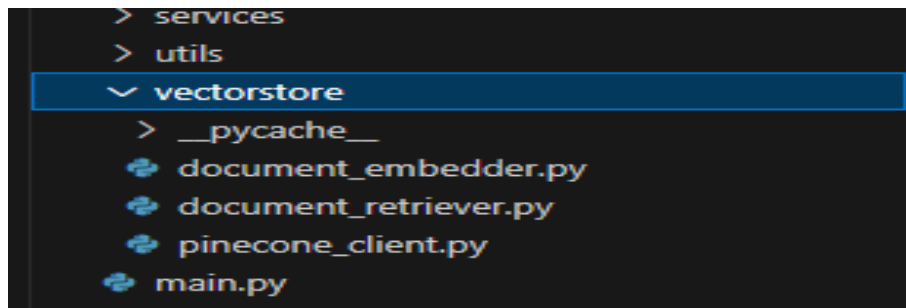
Icon-rich sidebar using streamlit-option-menu

Rounded buttons, font styles, padding fixes

Phase 5 – Pinecone & Document Embedding

Embedding Logic Built:

Created `document_embedder.py` and `document_retriever.py` using `sentence-transformers`.



Phase 6 – Report Generation & Deployment

Granite LLM Report Generator:

report_generator.py takes city name and KPI data, generates detailed city sustainability report using Granite LLM prompts.

Markdown & PDF Support:

Output formatted to text block for copy/paste or PDF download (optional).

End-to-End Integration Testing:

Final dashboard tested on all 8 features: KPI dashboard, feedback form, policy summarization, eco tips, chat, anomaly check, vector search, report generation.

Milestone 1: Requirements Specification

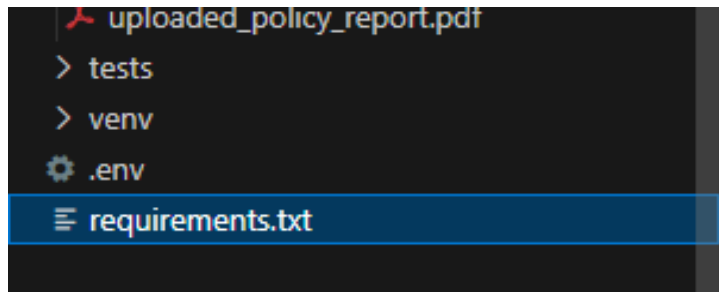
Objective: Establish the foundational libraries and packages for both frontend and backend to ensure reproducibility and easy environment setup.

Activity 1: Create requirements.txt

Duration: 0.5 Hrs

Skill Tags:

Activity 1: Create requirements.txt



Define the required libraries:

streamlit: For building interactive dashboard interfaces

fastapi: Backend API framework for rapid development

uvicorn: ASGI server to run FastAPI

requests: For API communication from frontend

python-dotenv: Manage environment variables

sentence-transformers: Text embedding model

pydantic-settings: Handle configuration management

pinecone-client: For semantic document search

scikit-learn, pandas: For anomaly detection and forecasting

matplotlib: For report visualization

Activity 2 Install all dependencies

Duration: 0.5 Hrs

Skill Tags:

bash

pip

install

-r

requirements.txt

```
requirements.txt
1 altair==5.5.0
2 annotated-types==0.7.0
3 anyio==4.9.0
4 asgiref==3.8.1
5 attrs==25.3.0
6 backoff==2.2.1
7 bcrypt==4.3.0
8 blinker==1.9.0
9 build==1.2.2.post1
10 cachetools==5.5.2
11 certifi==2025.1.31
12 charset-normalizer==3.4.1
13 chroma-hnswlib==0.7.6
14 chromadb==1.0.0
15 click==8.1.8
16 colorama==0.4.6
17 coloredlogs==15.0.1
18 Deprecated==1.2.18
19 distro==1.9.0
20 durationpy==0.9
```

Milestone 2: Environment Initialization & API Key Setup

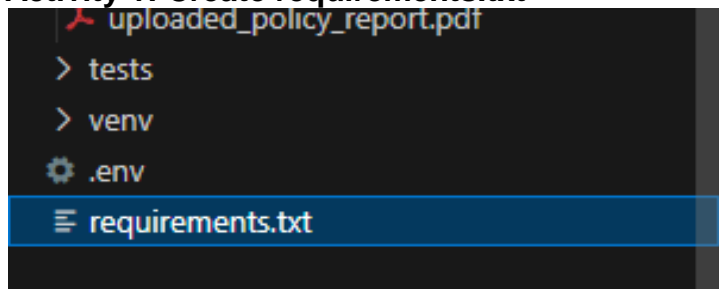
Objective: Configure and secure external service credentials (Watsonx & Pinecone).

Activity 1: Create requirements.txt

Duration: 0.5 Hrs

Skill Tags:

Activity 1: Create requirements.txt



```
uploaded_policy_report.pdf
> tests
> venv
⚙ .env
requirements.txt
```

Define the required libraries:

streamlit: For building interactive dashboard interfaces

fastapi: Backend API framework for rapid development

uvicorn: ASGI server to run FastAPI

: For API communication from frontend

python-dotenv: Manage environment variables

sentence-transformers: Text embedding model

pydantic-settings: Handle configuration management

pinecone-client: For semantic document search

scikit-learn, pandas: For anomaly detection and forecasting

matplotlib: For report visualizations

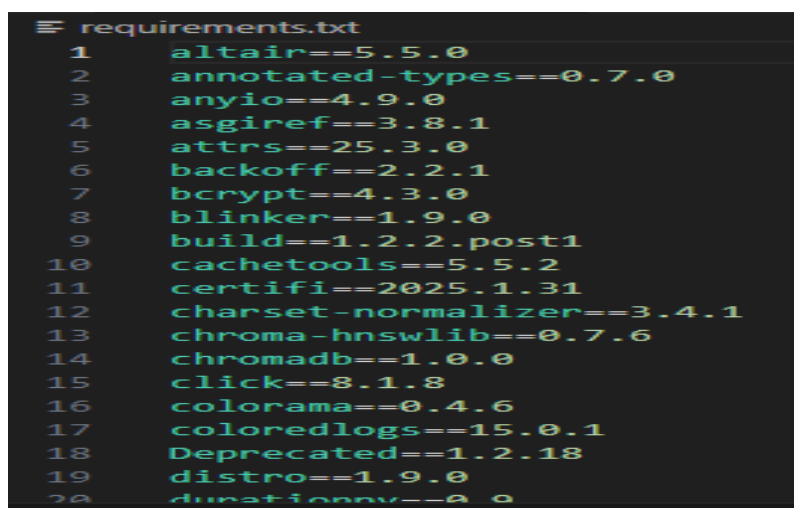
Activity 2: Install all dependencies

Duration: 0.5 Hrs

Skill Tags:

bash

pip install -r requirements.txt

A screenshot of a terminal window with a dark background. The title bar of the window reads "requirements.txt". The terminal displays a list of 20 dependencies, each on a new line, numbered 1 through 20. The dependencies are: altair==5.5.0, annotated-types==0.7.0, anyio==4.9.0, asgiref==3.8.1, attrs==25.3.0, backoff==2.2.1, bcrypt==4.3.0, blinker==1.9.0, build==1.2.2.post1, cachetools==5.5.2, certifi==2025.1.31, charset-normalizer==3.4.1, chroma-hnswlib==0.7.6, chromadb==1.0.0, click==8.1.8, colorama==0.4.6, coloredlogs==15.0.1, Deprecated==1.2.18, distro==1.9.0, and durationpy==0.9. The text is in a light-colored monospace font.

```
requirements.txt
1 altair==5.5.0
2 annotated-types==0.7.0
3 anyio==4.9.0
4 asgiref==3.8.1
5 attrs==25.3.0
6 backoff==2.2.1
7 bcrypt==4.3.0
8 blinker==1.9.0
9 build==1.2.2.post1
10 cachetools==5.5.2
11 certifi==2025.1.31
12 charset-normalizer==3.4.1
13 chroma-hnswlib==0.7.6
14 chromadb==1.0.0
15 click==8.1.8
16 colorama==0.4.6
17 coloredlogs==15.0.1
18 Deprecated==1.2.18
19 distro==1.9.0
20 durationpy==0.9
```

Milestone 3: AI Model Integration

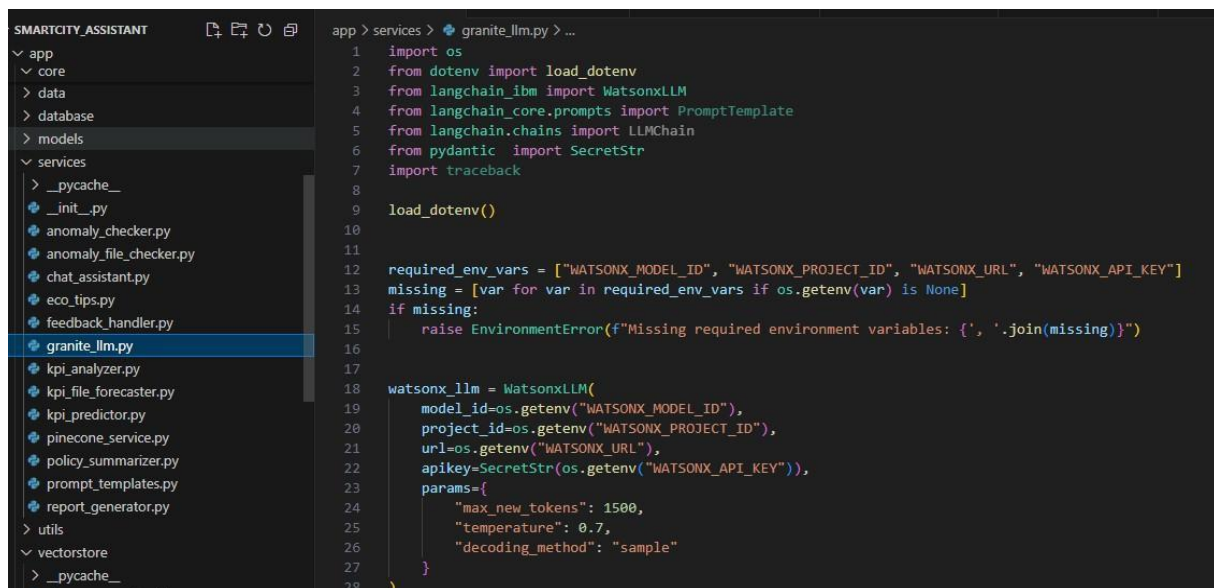
Objective: Integrate Watsonx Granite LLM with a centralized service layer.

Activity 1: Watsonx Integration

Duration: 0.5 Hrs

Skill Tags:

- Load env variables using python-dotenv
- Set up granite_llm.py to handle summarization, chat, eco tips, and sustainability reports
- Test LLM endpoints using dummy prompts



```
SMARTCITY_ASSISTANT
├── app
├── core
├── data
├── database
├── models
├── services
│   ├── __pycache__
│   ├── __init__.py
│   ├── anomaly_checker.py
│   ├── anomaly_file_checker.py
│   ├── chat_assistant.py
│   ├── eco_tips.py
│   ├── feedback_handler.py
│   └── granite_llm.py
│   ├── kpi_analyzer.py
│   ├── kpi_file_forecaster.py
│   ├── kpi_predictor.py
│   ├── pinecone_service.py
│   ├── policy_summarizer.py
│   ├── prompt_templates.py
│   └── report_generator.py
├── utils
├── vectorstore
├── __pycache__
└── ...

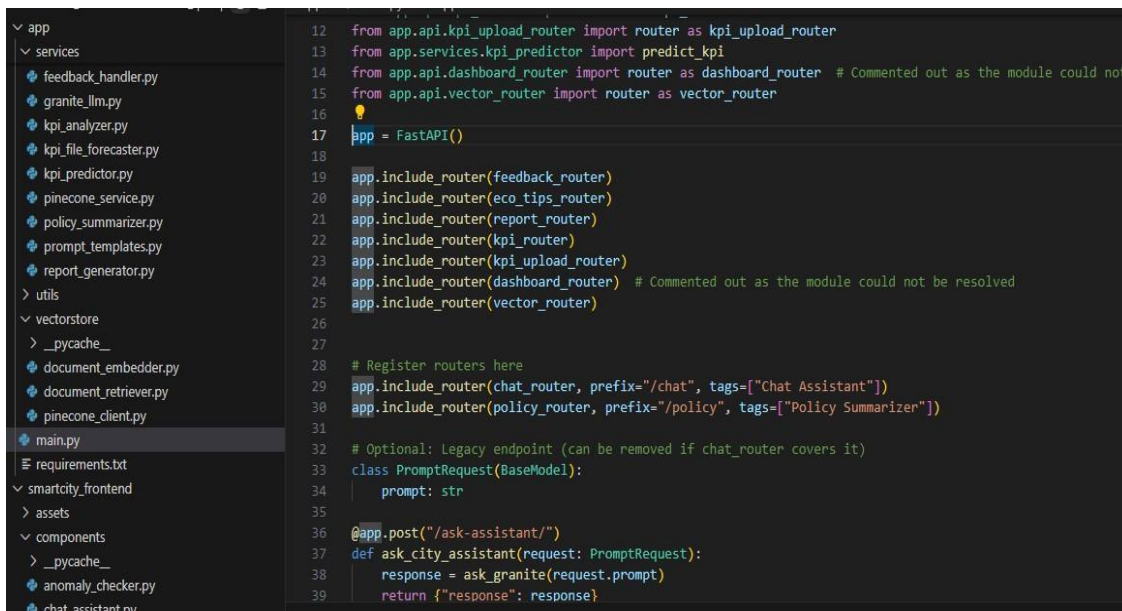
app > services > granite_llm.py > ...
1  import os
2  from dotenv import load_dotenv
3  from langchain_ibm import WatsonxLLM
4  from langchain_core.prompts import PromptTemplate
5  from langchain.chains import LLMChain
6  from pydantic import SecretStr
7  import traceback
8
9  load_dotenv()
10
11
12  required_env_vars = ["WATSONX_MODEL_ID", "WATSONX_PROJECT_ID", "WATSONX_URL", "WATSONX_API_KEY"]
13  missing = [var for var in required_env_vars if os.getenv(var) is None]
14  if missing:
15      raise EnvironmentError(f"Missing required environment variables: {' '.join(missing)}")
16
17
18  watsonx_llm = WatsonxLLM(
19      model_id=os.getenv("WATSONX_MODEL_ID"),
20      project_id=os.getenv("WATSONX_PROJECT_ID"),
21      url=os.getenv("WATSONX_URL"),
22      apikey=SecretStr(os.getenv("WATSONX_API_KEY")),
23      params={
24          "max_new_tokens": 1500,
25          "temperature": 0.7,
26          "decoding_method": "sample"
27      }
28  )
```

Activity 2: Implement LLM Service Functions

Duration: 0.5 Hrs

Skill Tags:

- ask_granite(prompt) for chat
- generate_summary(text) for policy summarization
- generate_eco_tip(topic) for environmental suggestions
- generate_city_report(kpi_data) for sustainability reports



The screenshot shows a code editor with a project structure on the left and Python code on the right. The project structure includes a 'services' folder with files like 'feedback_handler.py', 'granite_llm.py', 'kpi_analyzer.py', 'kpi_file_forecaster.py', 'kpi_predictor.py', 'pinecone_service.py', 'policy_summarizer.py', 'prompt_templates.py', and 'report_generator.py'. The main code file is 'main.py'. The Python code defines a FastAPI application, imports various routers, and includes a POST endpoint for '/ask-assistant/' that calls 'ask_granite'.

```
12 from app.api.kpi_upload_router import router as kpi_upload_router
13 from app.services.kpi_predictor import predict_kpi
14 from app.api.dashboard_router import router as dashboard_router # Commented out as the module could not
15 from app.api.vector_router import router as vector_router
16
17 app = FastAPI()
18
19 app.include_router(feedback_router)
20 app.include_router(eco_tips_router)
21 app.include_router(report_router)
22 app.include_router(kpi_router)
23 app.include_router(kpi_upload_router)
24 app.include_router(dashboard_router) # Commented out as the module could not be resolved
25 app.include_router(vector_router)
26
27
28 # Register routers here
29 app.include_router(chat_router, prefix="/chat", tags=["Chat Assistant"])
30 app.include_router(policy_router, prefix="/policy", tags=["Policy Summarizer"])
31
32 # Optional: Legacy endpoint (can be removed if chat_router covers it)
33 class PromptRequest(BaseModel):
34     prompt: str
35
36 @app.post("/ask-assistant/")
37 def ask_city_assistant(request: PromptRequest):
38     response = ask_granite(request.prompt)
39     return {"response": response}
```

Milestone 4: Backend API Development

Objective: Build modular RESTful API routes using FastAPI.

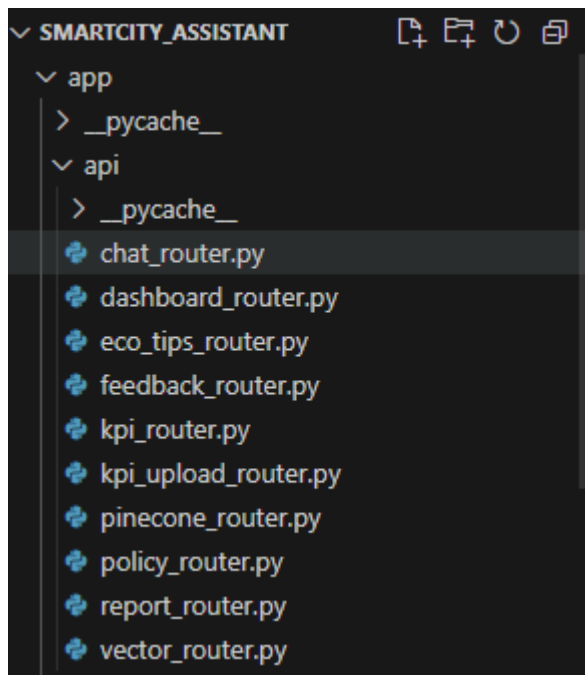
Activity 1: Create Routers

Duration: 0.5 Hrs

Skill Tags:

Modules created in app/api/:

- chat_router.py
- policy_router.py
- eco_tips_router.py
- feedback_router.py
- report_router.py
- vector_router.py
- kpi_upload_router.py
- dashboard_router.py



Activity 2: Test Routes

Duration: 0.5 Hrs

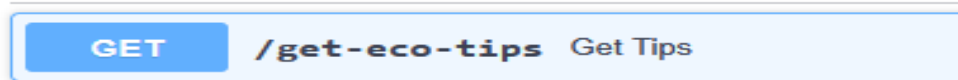
Skill Tags:

Use Swagger UI to validate:

- POST /upload-doc

- GET /search-docs
- GET /get-eco-tips?topic=energy
- POST /submit-feedback

Citizen Tips



Milestone 5: Streamlit Frontend UI Development

Objective: Design a user-friendly dashboard for real-time interaction.

Activity 1: Page Structure

Duration: 0.5 Hrs

Skill Tags:

- Sidebar navigation using streamlit-option-menu
- Separate pages for: Dashboard, Feedback, Eco Tips, Chat, Policy Search, Anomaly Checker, KPI Forecasting

Milestone 6: Pinecone Semantic Search Integration

Objective: Embed uploaded documents and enable semantic policy search. **Document Embedding**

- Use sentence-transformers model (MiniLM) to convert .txt into 384-d vectors
- Store documents in Pinecone via document_embedder.py

Vector Search

POST	/upload-doc	Upload Doc
GET	/search-docs	Search Docs

Milestone 7: ML-based Forecasting and Anomaly Detection

Objective: Analyze uploaded CSV files and predict future trends or irregularities.

Activity 1: Forecasting

Duration: 0.5 Hrs


Skill Tags:

- Use Linear Regression in `kpi_file_forecaster.py`
- Predict water/energy use based on past data
- Display forecast on dashboard

```

app > services > kpi_file_forecaster.py > ...
1  import pandas as pd
2  from sklearn.linear_model import LinearRegression
3  from io import StringIO
4
5  def forecast_from_uploaded_csv(content: str, kpi: str) -> dict:
6      df = pd.read_csv(StringIO(content))
7
8      if 'year' not in df.columns or kpi not in df.columns:
9          return {"error": "Missing 'year' or KPI column in uploaded file."}
10
11     model = LinearRegression()
12     model.fit(df[['year']], df[['kpi']])
13
14     next_year = df['year'].max() + 1
15     prediction = model.predict(pd.DataFrame([[next_year]], columns=["year"]))[0]
16
17
18     return {
19         "predicted_year": int(next_year),          # Cast to int
20         "kpi": str(kpi),                          # Ensure string
21         "predicted_value": float(round(prediction[0], 2)) # Access first element from NumPy array
22     }
23


```



KPI Forecasting


Upload a `.csv` file containing historical KPI data (e.g., energy, water, waste). The model will predict the next year's value.

Upload CSV File



Drag and drop file here
 Limit 200MB per file • CSV

Browse files




delhi_1.csv 142.0B
 ×

Select KPI to forecast

energy

▼


Predict KPI

Debug Response:

Activity 2: Anomaly Detection

Duration: 0.5 Hrs

Skill Tags:

- `anomaly_file_checker.py` flags abnormal spikes


- Display results in tabular or colored badge format

```
app > services > anomaly_file_checker.py > ...
1 import pandas as pd
2 from io import StringIO
3
4 def detect_anomaly_in_uploaded_csv(content: str, kpi: str, threshold: float) -> dict:
5     df = pd.read_csv(StringIO(content))
6
7     if kpi not in df.columns:
8         return {"error": f"KPI '{kpi}' not found in file."}
9
10    anomalies = df[df[kpi] > threshold]
11
12    return {
13        "kpi": kpi,
14        "threshold": threshold,
15        "anomaly_count": len(anomalies),
16        "anomalies": anomalies.to_dict(orient="records")
17    }
```


Anomaly Detection

Upload a `.csv` file with KPI data to check for anomalies based on a threshold value.

Upload CSV File

 Drag and drop file here
Limit 200MB per file • CSV

Browse files


 delhi_1.csv 142.0B ×

Select KPI to check

energy ▼

Set Threshold Value

1000.00 - +

 Check for Anomalies

5 anomaly/anomalies found:

Milestone 8: Sustainability Report Generation

Objective: Generate a city-wise AI-powered sustainability summary.

Activity 1: Prompt Engineering

Duration: 0.5 Hrs

Skill Tags:

- `report_generator.py` uses a custom prompt to generate an AI-written report from KPI inputs

Activity 2: Display/Download

Duration: 0.5 Hrs

Skill Tags:

- Render AI report on frontend
- Optionally provide markdown/PDF output

```
app > services > policy_summarizer.py > ...
1  from app.services.granite_llm import ask_granite
2
3  def summarize_policy(text: str) -> str:
4      prompt = (
5          "You are a smart city assistant AI. Read the following policy text carefully, "
6          "and generate a concise summary of minimum 50 words using simple language for citizens to understand."
7          f"Policy Text:\n{text}\n\nSummary:"
8      )
9      return ask_granite(prompt)
10
```



Policy Summarizer

Choose input method



Enter Text



Upload .txt File

Paste your policy or city guideline below



Summarize Text




Download Sustainability Report (PDF)


```

app > services > report_generator.py > ...
1  from app.services.granite_llm import ask_granite
2  import os
3  from datetime import datetime
4  from pathlib import Path
5  from fpdf import FPDF
6
7  def generate_sustainability_report(content: str) -> str:
8      prompt = (
9          "You are a sustainability analyst. Based on the following document or met
10         "generate a concise sustainability report suitable for city planners:\n\n
11         f"{content}\n\n"
12         "Structure the report with clear headings like Summary, Challenges, Recom
13     )
14     return ask_granite(prompt)
15
16 def generate_markdown_report(summary: str) -> str:
17     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
18     filename = f"sustainability_report_{timestamp}.md"
19     path = Path("app/data") / filename
20
21     with open(path, "w", encoding="utf-8") as f:
22         f.write("# Sustainability Report\n\n")
23         f.write(summary)
24


```



Sustainability Report Generator


Upload a `.txt` file containing policy or city metric data. The assistant will generate a downloadable PDF report.


Upload Policy Text File (.txt)




Drag and drop file here
 Limit 200MB per file • TXT

Browse files


smartcity_1.txt 1.0KB
×


Generate Report

Report generated successfully!


Download PDF Report

Milestone 9: Chat Assistant Creation

Objective: Build an interactive chat module where users can ask AI-driven questions related to sustainability, city governance, and smart living—powered by IBM Watsonx Granite LLM.

Activity 1: Define Backend Route

Duration: 0.5 Hrs

Skill Tags:

- Create `chat_router.py` in the `app/api/` directory.
- Endpoint `/chat/ask` accepts a prompt string as input.
- Calls the `ask_granite()` function from `granite_llm.py` to generate the response.

```

app > services > chat_assistant.py > ...
1  from app.services.granite_llm import ask_granite
2
3  def ask_city_assistant(prompt: str) -> str:
4      system_message = (
5          "You are a helpful assistant who answers detailed questions related to smart cities, "
6          "sustainability, green infrastructure, and pollution reduction. Provide clear, complete, and structured responses."
7      )
8      full_prompt = f"<system>: {system_message}\n<user>: {prompt}\n<assistant>:"
9      return ask_granite(full_prompt)
10

```



Chat with Smart City Assistant

Ask anything about sustainability, smart policies, green infrastructure, or city planning!

Your question

Send

You: how to reduce air pollution?

Assistant: Use public transportation more often.

You: how to reduce air pollution. give 5 ways?

Assistant: 1. Use public transportation more often

You: how to reduce air pollution. give "5" ways?

Assistant: 1. Plant more trees 2. Use public transport 3. Walk or cycle more 4. Buy an electric car

Milestone 10: Final Integration & Testing

Objective: Ensure smooth interaction across modules.

Activity 1: Connect All Pages

Duration: 0.5 Hrs

Skill Tags:

- Navigation working via sidebar

- Real-time API interactions tested

Activity 2: Run Final Test

Duration: 0.5 Hrs

Skill Tags:

uvicorn app.main:app --reload streamlit run smart_dashboard.py

```
(venv) PS C:\Users\Admin\Desktop\Smartbridge\Smartcity_assistant> uvicorn app.main:app --reload
>>
INFO: Will watch for changes in these directories: ['C:\\Users\\Admin\\Desktop\\Smartbridge\\Smartcity_assistant']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [4468] using WatchFiles
INFO: Started server process [11772]
INFO: Waiting for application startup.
INFO: Application startup complete.

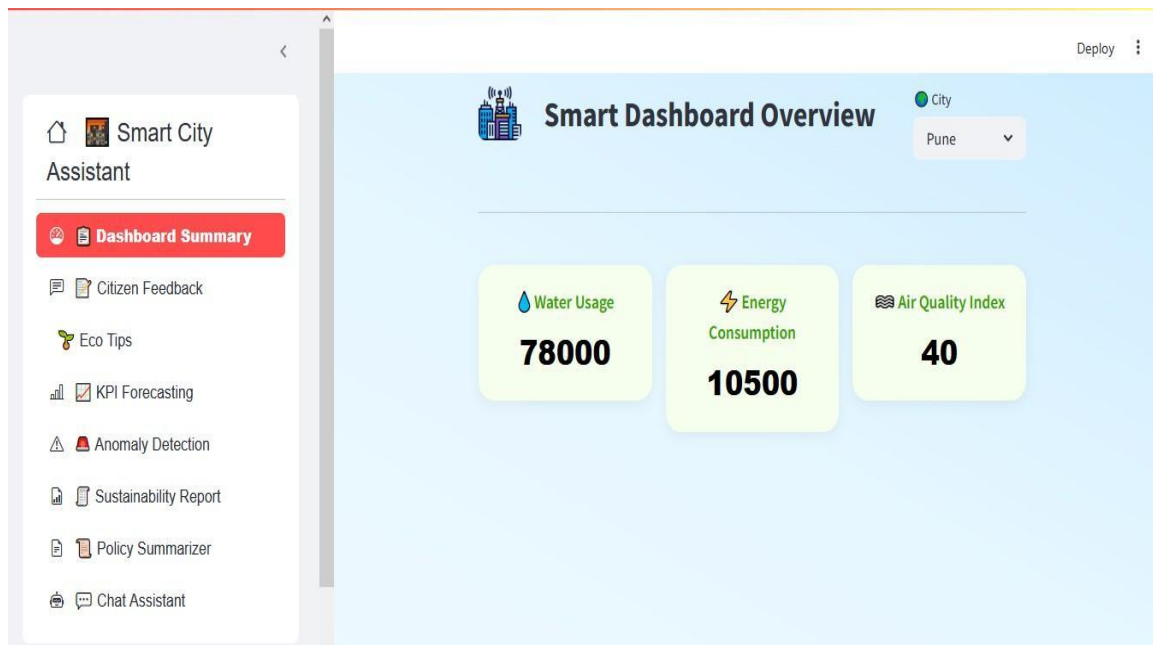
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
INFO: 127.0.0.1:54767 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54762 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54763 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54767 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54763 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54767 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54767 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54833 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:54834 - "GET /dashboard/summary?city=Pune HTTP/1.1" 200 OK
INFO: 127.0.0.1:50430 - "POST /forecast-from-file?kpi=energy HTTP/1.1" 200 OK
INFO: 127.0.0.1:50448 - "POST /check-anomaly-from-file?kpi=energy&threshold=1000.0 HTTP/1.1" 200 OK
INFO: 127.0.0.1:50473 - "POST /policy/generate-report HTTP/1.1" 200 OK

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Admin\Desktop\Smartbridge\Smartcity_assistant> cd smartcity_frontend
PS C:\Users\Admin\Desktop\Smartbridge\Smartcity_assistant\smartcity_frontend> streamlit run smart_dashboard.py
>>

You can now view your Streamlit app in your browser.

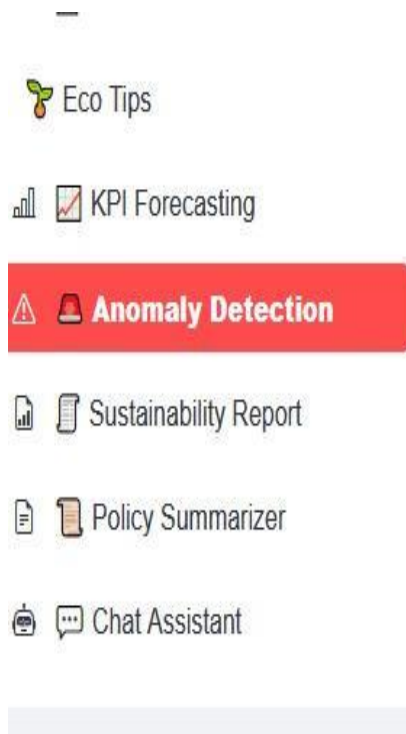
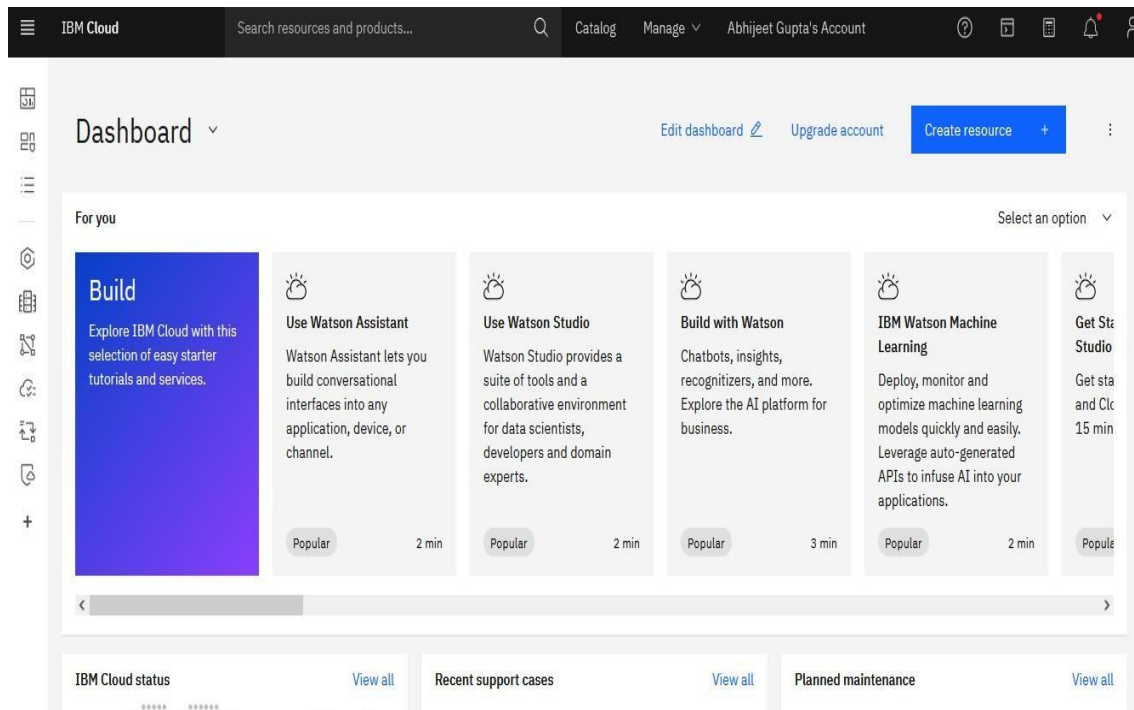
Local URL: http://localhost:8501
Network URL: http://192.168.1.43:8501
```

Screenshots / Outputs:



Policy Summarizer

POST	/policy/summarize-policy	Summarize
GET	/policy/test-llm	Test Llm
GET	/policy/summarize-from-file	Summarize From File
POST	/policy/summarize-uploaded-file	Summarize Uploaded File
POST	/policy/generate-markdown-report	Generate Md From Text
POST	/policy/generate-pdf-report	Generate Pdf From Text
POST	/policy/upload-txt-generate-markdown	Generate Md From Up
POST	/policy/upload-txt-generate-pdf	Generate Pdf From Uploaded
POST	/policy/generate-report	Generate Report From Uploaded Txt




```
app > data > {} feedback_log.json > {} 7
```

```
1  [
2    {
3      "timestamp": "2025-04-09T14:20:13.013642",
4      "user": "abhiijeet",
5      "category": "waste management",
6      "message": "Garbage bins in sector 5 are overflowing every day."
7    },
8    {
9      "timestamp": "2025-04-10T07:01:13.630209",
10     "user": "Vaisali",
11     "category": "Energy",
12     "message": "streets lights are On during Daylight"
13   },
14   {
15     "timestamp": "2025-04-10T11:51:23.871276",
16     "user": "Ankit",
17     "category": "Water",
18     "message": "water supply not available in sector 5 . noida"
19   },
20   {
```



Eco-Friendly Tips Assistant

Get actionable suggestions to live sustainably based on your query.

 Enter a topic (e.g., recycling, energy, water, plastic):

green park . "10" tips

 Get Eco Tip



Tip: 1. Use public transportation more often. 2. Walk or bike more often. 3. Buy products that are produced in an environmentally friendly way. 4. Recycle whenever possible. 5. Turn off lights when not in use. 6. Use energy-efficient light bulbs. 7. Use water-saving showerheads and faucet aerators. 8. Use a water filter instead of buying bottled water. 9. Reduce food waste. 10. Support businesses that are committed to sustainability.



Predict KPI

Debug Response:

```
{  
  "predicted_year" : 2024  
  "kpi" : "energy"  
  "predicted_value" : 1528.67  
}
```

Forecasted ENERGY for next year: 1528.67