**Project Title**

**Project Documentation**

# 1.Introduction

* Project title : CITIZENS AI ENGAGEMENT
* Team member : Arivuselvi.P
* Team member : Bhuvaneswari.D
* Team member : Archana.R
* Team member : Arthi.R

# 2.project overview

* **Purpose** :
* To enhance citizen–government engagement by providing real-time AI assistance, analyzing public sentiment, and delivering actionable insights through dynamic dashboards—ultimately improving service delivery, efficiency, and trust in governance.
* **Features**:

1. **Conversational Interface**

Key Point: Natural language interaction

Functionality: Citizens/officials ask questions, get updates, and receive guidance in plain language.

1. **Policy Summarization**

Key Point: Simplified policy understanding

Functionality: Converts lengthy government documents into concise, actionable summaries.

1. **Resource Forecasting**

Key Point: Predictive analytics

Functionality: Estimates future energy, water, and waste usage from historical + real-time data.

1. **Eco-Tip Generator**

Key Point: Personalized sustainability advice

Functionality: Suggests daily eco-friendly actions based on user behavior.

1. **Citizen Feedback Loop**

Key Point: Community engagement

Functionality: Collects and analyzes public input to improve city planning and services.

1. **KPI Forecasting**

Key Point: Strategic planning support

Functionality: Projects KPIs for tracking progress and guiding future decisions.

1. **Anomaly Detection**

Key Point: Early warning system

Functionality: Detects unusual data patterns to flag issues early (e.g., service disruptions).

1. **Multimodal Input Support**

Key Point: Flexible data handling

Functionality: Handles text, PDFs, and CSVs for document analysis and forecasting.

1. **Streamlit/Gradio UI**

Key Point: User-friendly dashboard

Functionality: Intuitive interface for both citizens and officials to interact with the assistant.

# 3. Architecture

**Frontend (Streamlit):**

* Built with Streamlit for an interactive multi-page web UI.
* Features: dashboards, file uploads, chat interface, feedback forms, report viewers.
* Sidebar navigation with streamlit-option-menu.
* Modularized pages → scalable and easy to maintain.

**Backend (FastAPI):**

* Powered by FastAPI REST framework.
* Provides endpoints for: document processing, chat, eco-tip generation, report creation, vector embedding.
* Optimized for asynchronous performance with easy Swagger UI integration.
* LLM Integration (IBM Watsonx Granite):
* Uses Granite LLMs for natural language understanding and generation.
* Prompt engineering enables: summaries, sustainability tips, and reports.

**LLM Integration (IBM Watsonx Granite):**

* Uses Granite LLMs for natural language understanding and generation.
* Prompt engineering enables: summaries, sustainability tips, and reports.

**Vector Search (Pinecone):**

* Policy documents embedded with Sentence Transformers.
* Stored in Pinecone for semantic search.
* Cosine similarity used for natural language policy/document queries.
* ML Modules (Forecasting & Anomaly Detection):
* Built with Scikit-learn for lightweight ML.
* Forecasting and anomaly detection applied to time-series KPI data.
* Pandas + matplotlib used for modeling and visualization.

**ML Modules (Forecasting & Anomaly Detection):**

* Built with Scikit-learn for lightweight ML.
* Forecasting and anomaly detection applied to time-series KPI data.
* Pandas + matplotlib used for modeling and visualization.

**4. Setup Instructions**

**Prerequisites:**

* Python 3.9+
* Pip and virtual environment tools
* API keys for IBM Watsonx and Pinecone
* Stable internet access for 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 modules (chat, forecasting, eco-tips, reports)

## 5. Folder Structure

* App/ → FastAPI backend logic (routers, models, integrations).
* App/api/ → modular API routes (chat, feedback, report, document vectorization).
* Ui/ → Streamlit frontend components (pages, card layouts, form UIs).
* Smart\_dashboard.py → entry script to launch the main Streamlit dashboard.
* Granite\_llm.py → manages communication with IBM Watsonx Granite (summarization + chat).
* Document\_embedder.py → converts documents into embeddings, stores them in Pinecone.
* Kpi\_file\_forecaster.py → forecasts future energy/water trends with regression models.
* Anomaly\_file\_checker.py → detects unusual values in uploaded KPI data.
* Report\_generator.py → builds AI-generated sustainability reports.

6. **Running the Application**

**Steps to Start:**

* Launch the FastAPI server → exposes backend endpoints.
* Run the Streamlit dashboard → provides web interface.
* Use the sidebar navigation → switch between pages.
* Upload documents/CSVs, chat with the assistant, and view reports/summaries/predictions.
* All interactions happen in real-time, with backend APIs updating the frontend dynamically.

**Frontend (Streamlit):**

* Built with Streamlit, offering an interactive multi-page UI.
* Features: dashboards, file uploads, chat interface, feedback forms, report viewers.
* Sidebar navigation with streamlit-option-menu.
* Modularized pages → scalable and easy to extend.

**Backend (FastAPI):**

* Powered by FastAPI REST framework.
* Handles: document processing, chat interactions, eco-tip generation, report creation, vector embedding.
* Optimized for asynchronous performance.
* Integrated with Swagger UI for easy API testing.

## 7. API Documentation

* **POST /chat/ask** → accepts a user query and returns an AI-generated response.
* **POST /upload-doc** → uploads and embeds documents into Pinecone.
* **GET /search-docs** → retrieves semantically similar policies to the input query.
* **GET /get-eco-tips** → provides sustainability tips (energy, water, waste).
* **POST /submit-feedback** → stores citizen feedback for review and analytics.

. **8. Authentication**

* **Swagger UI** → each endpoint tested and documented for quick inspection/trials.
* **Current setup** → open environment for demo purposes.
* Secure deployment options:
* Token-based authentication (JWT / API keys)
* OAuth2 with IBM Cloud credentials
* Role-based access (admin, citizen, researcher)
* Planned enhancements: user sessions and history tracking.

## 9. User Interface

* **Minimalist & functional** → easy for non-technical users.
* **Sidebar navigation** → quick access to all modules.
* **KPI visualizations** → summary cards for key insights.
* **Tabbed layout** → chat, eco tips, forecasting.
* **Real-time form handling** → smooth data input.
* **PDF report download** → easy report generation.
* **Design focus** → clarity, speed, and user guidance with help texts & intuitive flows.

## 10. Testing

* Testing for Citizen AI was carried out in multiple phases to ensure robustness and reliability. Unit testing was first conducted on prompt engineering functions and utility scripts to validate their correctness at the component level. API testing was performed using Swagger UI, Postman, and automated test scripts to verify endpoint functionality and integration accuracy.
* Manual testing was then applied to evaluate file uploads, conversational responses, and overall output consistency from the user perspective. In addition, edge case handling was thoroughly examined by providing malformed inputs, large file uploads, and invalid API keys to test system stability under stress.
* Across all phases, each function was validated to perform reliably in both offline and API-connected modes, ensuring that Citizen AI delivers a consistent and dependable experience.

**11.screen shots**

1. **Known Issues**
2. **Future enhancement**