**Project Title**

**Project Documentation**

# 1.Introduction

* Project title : GENERATIVE AI WITH IBM
* Team member : ATCHAYA.K
* Team member : KARTHIKA .S
* Team member : POWSIYA.A
* Team member : LUXCI VALANTINA.B

# 2.project overview

* **Purpose** :
* Generative AI with IBM is designed to help businesses create new content, ideas, and solutions by using advanced AI models. Its purpose is to boost productivity, improve decision-making, automate tasks, enhance customer experiences, and support innovation across industries.
* **Features**:

**Content Creation – Generates text, code, images, and insights.**

**Automation** – Speeds up repetitive tasks and workflows.

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

**Integration** – Works with IBM tools like Watsonx and cloud services.

**Scalability** – Handles enterprise-level data and workloads.

**Trust** **&** **Security** – Focus on responsible, ethical, and secure AI.

# 3. Architecture

* **. Data Layer**
* Collects structured (databases, spreadsheets) and unstructured (documents, images, logs) data.
* Ensures data quality, governance, and compliance.
* Uses IBM watsonx.data for managing and scaling enterprise data.
* **2. Foundation Models**
* Large pre-trained AI models (LLMs, multimodal, code models).
* Trained on diverse datasets for general knowledge.
* IBM integrates open-source models + proprietary models.
* **3. Training & Fine-Tuning Layer**
* Enterprises fine-tune foundation models with their own data.
* Uses techniques like transfer learning, prompt-tuning, and supervised fine-tuning.
* Reduces cost/time compared to training from scratch.
* **4. Watsonx.ai Platform (Core AI Studio)**
* Central workspace for building, testing, and deploying generative AI solutions.
* Supports natural language processing, automation, chatbots, summarization, coding assistance, etc.
* Provides a low-code/no-code interface for easy use.
* 5**. Integration & API Layer**
* Connects AI outputs with enterprise applications (ERP, CRM, HR, finance tools, etc.).
* Provides APIs, SDKs, and connectors for cloud and on-prem systems.
* Enables embedding AI into workflows, mobile apps, and websites.

6. **Governance** & **Security** **Layer**

* Managed through watsonx.governance.
* Ensures transparency, explainability, fairness, and bias detection.
* Provides model monitoring, auditing, and compliance with global AI regulations.

**7**. **Deployment** **Layer**

* Flexible deployment: IBM Cloud, hybrid cloud, or on-premises.
* Scales according to business workload.
* Continuous monitoring & updates to improve reliability.

**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**