SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation

1.Introduction

- Project title : SmartSDLC – AI-Enhanced Software Development Lifecycle
- Team Leader /ID: marimuthu A /NM2025TMID07058
- Team member :Murugesan N
- Team member : CHANDHIRAN S
- Team member :Thirumoorthi T

2.project overview

- Purpose :

The purpose of a Sustainable 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 decisionmaking 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

Key Point: Natural language interaction

Functionality: Allows citizens and officials to ask questions, get updates, and receive guidance in plain language

Policy Summarization

Key Point: Simplified policy understanding

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

Resource Forecasting

Key Point: Predictive analytics

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

Eco-Tip Generator

Key Point: Personalized sustainability advice

Functionality: Recommends daily actions to reduce environmental impact based on user behavior.

Citizen Feedback Loop

Key Point: Community engagement

Functionality: Collects and analyzes public input to inform city planning and service improvements.

KPI Forecasting

Key Point: Strategic planning support

Functionality: Projects key performance indicators to help officials track progress and plan ahead.

Anomaly Detection

Key Point: Early warning system

Functionality: Identifies unusual patterns in sensor or usage data to flag potential issues.

Multimodal Input Support

Key Point: Flexible data handling

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

Streamlit or Gradio UI

Key Point: User-friendly interface

Functionality: Provides an intuitive dashboard for both citizens and city officials to interact with the assistant.

3. Architecture

Frontend (Stream lit):

The frontend is built with Stream lit, offering an interactive web UI with multiple pages including dashboards, file uploads, chat interface, feedback forms, and report viewers. Navigation is handled through a sidebar using the stream lit-option-menu library. Each page is modularized for scalability.

Backend (Fast API):

Fast API serves as the backend REST framework that powers API endpoints for document processing, chat interactions, eco tip generation, report creation, and vector embedding. It is optimized for asynchronous performance and easy Swagger integration.

LLM Integration (IBM Watsonx Granite):

Granite LLM models from IBM Watsonx are used for natural language understanding and generation. Prompts are carefully designed to generate summaries, sustainability tips, and

reports.

Vector Search (Pinecone):

Uploaded policy documents are embedded using Sentence Transformers and stored in Pinecone. Semantic search is implemented using cosine similarity to allow users to search documents using natural language queries.

ML Modules (Forecasting and Anomaly Detection):

Lightweight ML models are used for forecasting and anomaly detection using Scikit-learn. Time-series data is parsed, modeled, and visualized using pandas and matplotlib. **4. 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 Fast API ○ Launch the frontend via Stream lit ○ Upload data and interact with the modules

5. Folder Structure

app/ – Contains all Fast API backend logic including routers, models, and integration modules.

app/api/ – Subdirectory for modular API routes like chat, feedback, report, and document vectorization.

ui/ – Contains frontend components for Stream lit pages, card layouts, and form UIs.

smart_dashboard.py – Entry script for launching the main Stream lit dashboard.

granite_llm.py – Handles all communication with IBM Watsonx Granite model including summarization and chat.

document_embedder.py – Converts documents to embeddings and stores in Pinecone.

kpi_file_forecaster.py – Forecasts future energy/water trends using regression.

anomaly_file_checker.py – Flags unusual values in uploaded KPI data. report_generator.py – Constructs AI-generated sustainability reports.

6. Running the Application

To start the project:

- Launch the FastAPI server to expose backend endpoints.
- Run the Streamlit dashboard to access the web interface.
- Navigate through pages via the sidebar.
- Upload documents or CSVs, interact with the chat assistant, and view

outputs like reports, summaries, and predictions.

- All interactions are real-time and use backend APIs to dynamically update the frontend.

Frontend (Stream lit):

The frontend is built with Stream lit, offering an interactive web UI with multiple pages including dashboards, file uploads, chat interface, feedback forms, and report viewers. Navigation is handled through a sidebar using the stream lit-option-menu library. Each page is modularized for scalability.

Backend (Fast API):

Fast API serves as the backend REST framework that powers API endpoints for document processing, chat interactions, eco tip generation, report creation, and vector embedding. It is optimized for asynchronous performance and easy Swagger integration.

7. API Documentation

Backend APIs available include:

POST /chat/ask – Accepts a user query and responds with an AI-generated message

POST /upload-doc – Uploads and embeds documents in Pinecone

GET /search-docs – Returns semantically similar policies to the input query

GET /get-eco-tips – Provides sustainability tips for selected topics like energy, water, or waste

POST /submit-feedback – Stores citizen feedback for later review or analytics

Each endpoint is tested and documented in Swagger UI for quick inspection and trial during development. **8. Authentication**

each endpoint is tested and documented in Swagger UI for quick inspection and trial during development.

This version of the project runs in an open environment for demonstration.

However, secure deployments can integrate:

- Token-based authentication (JWT or API keys)
- OAuth2 with IBM Cloud credentials
- Role-based access (admin, citizen, researcher)
- Planned enhancements include user sessions and history tracking.8. Authentication

9. User Interface

The interface is minimalist and functional, focusing on accessibility for nontechnical users. It includes:

- Sidebar with navigation
- KPI visualizations with summary cards
- Tabbed layouts for chat, eco tips, and forecasting
- Real-time form handling
- PDF report download capability

The design prioritizes clarity, speed, and user guidance with help texts and intuitive flows.

10. Testing

Testing was done in multiple phases:

Unit Testing: For prompt engineering functions and utility scripts

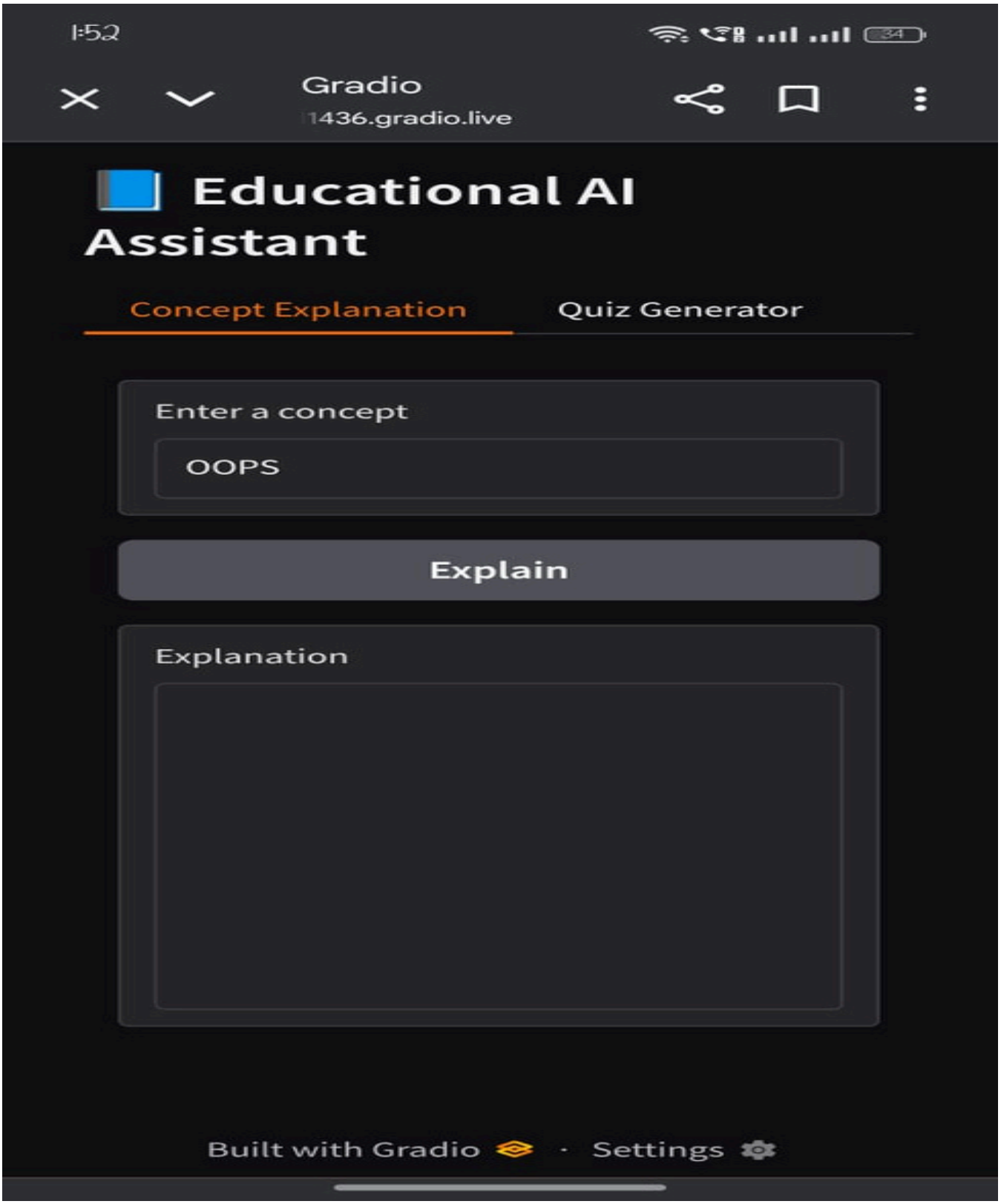
API Testing: Via Swagger UI, Postman, and test scripts

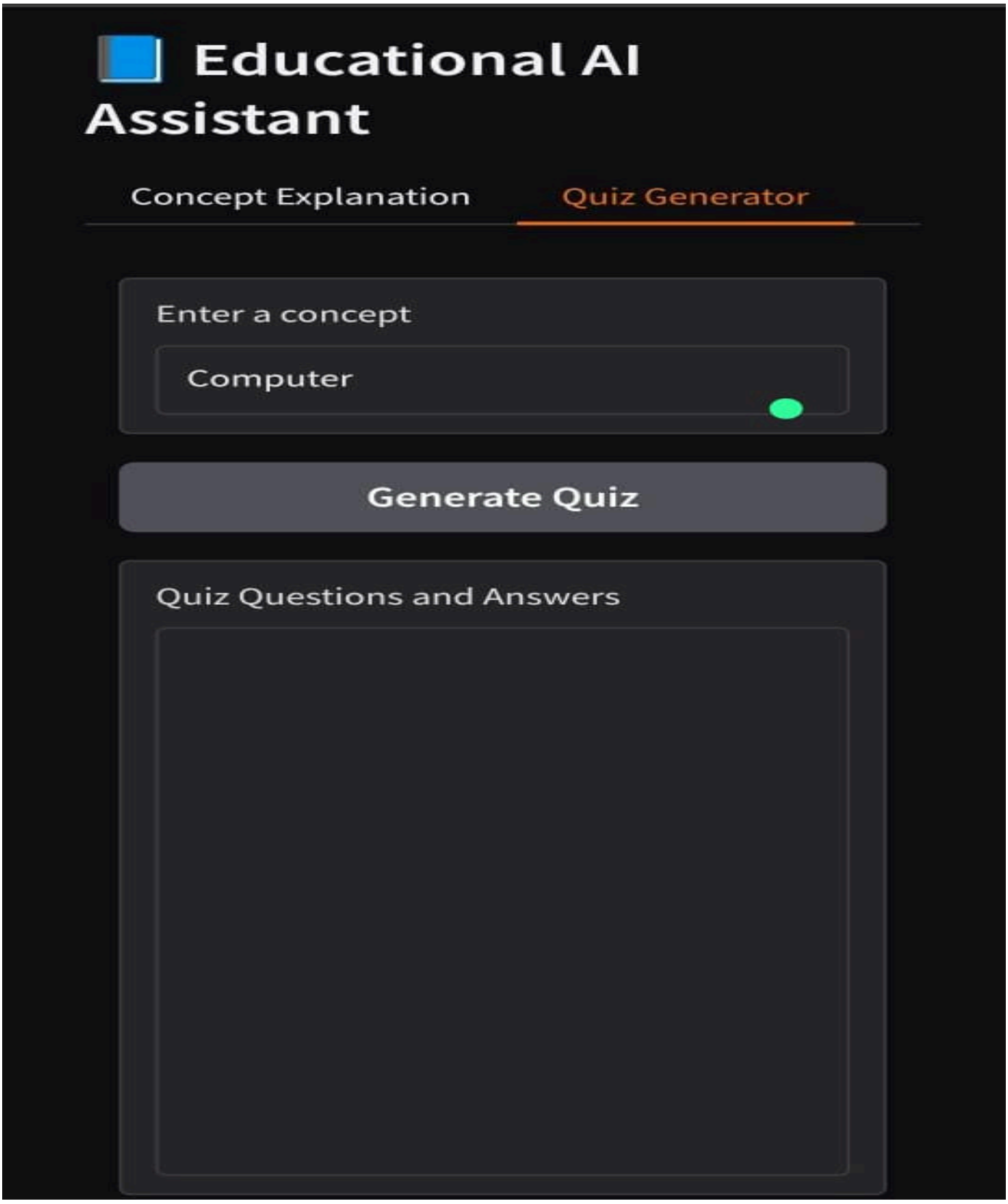
Manual Testing: For file uploads, chat responses, and output consistency

Edge Case Handling: Malformed inputs, large files, invalid API keys

Each function was validated to ensure reliability in both offline and APIconnected modes.

11.screen shots


1:52
Gradio
1436.gradio.live
Educational AI Assistant
Concept Explanation
Quiz Generator
Enter a concept
OOPS
Explain
Explanation
Built with Gradio
Settings



1. **Known Issues**

- 1. Data Dependency & Quality
- AI models need huge amounts of high-quality training data.
- Poor, biased, or incomplete datasets can lead to wrong predictions in requirement analysis, testing, or security scanning.
- 2. Security Risks

- AI-driven code generation may unintentionally introduce hidden vulnerabilities.
- Malicious actors may exploit AI tools in DevOps pipelines.
- Data privacy issues when using AI-based requirement gathering from client documents/emails.
- 3. Bias & Ethical Concerns
- AI systems may produce biased designs or decisions due to skewed training data.
- Risk of unfair task allocation or biased bug prioritization.
- Ethical questions: Who is accountable for AI-driven errors?
- 4. Over-Reliance on AI
- Developers may become overdependent on AI-driven coding tools.
- Creativity, problem-solving, and algorithmic thinking may decrease in human engineers.
- Risk of "black-box" decisions where humans don't understand AI reasoning.
- 5. Integration Challenges
- Difficulty integrating AI tools with existing legacy systems.
- Lack of standard frameworks for AI-driven SDLC.
- High cost of adoption for small and medium enterprises (SMEs).
- 6. Complexity in Debugging AI-Generated Code
- AI-generated code may be efficient but hard to debug due to lack of clarity.
- Traceability and accountability issues: Who fixes an AI-written bug?
- 7. Resource Intensive
- Training and running AI models require high computing power (GPUs, cloud).
- Increased energy consumption, raising sustainability concerns.
- 8. Continuous Model Maintenance
- AI models need constant retraining to stay updated with new programming languages, frameworks, and security threats.
- Maintenance of AI models may become more complex than traditional SDLC.
- 9. Legal & Compliance Issues
- AI in software may face regulatory challenges (e.g., GDPR, copyright in AI-generated code).

2. Future enhancement

- Future Enhancements of Smart SDLV
- 1. Requirement Analysis & Planning
- AI-based requirement gathering from natural language documents, emails, and client conversations.
- Predicting feasibility and project risks using predictive analytics.
- AI-driven cost and time estimation models.
- 2. System Design
- AI-generated design blueprints and automated UML diagrams.
- Intelligent suggestion of best design patterns.
- Adaptive architecture modeling based on project needs.
- 3. Coding / Implementation
- AI-driven code generation (e.g., GitHub Copilot-like systems).
- Automatic code refactoring and bug detection.
- Self-healing code that fixes detected vulnerabilities automatically.
- 4. Testing
- AI-based test case generation and prioritization.
- Predictive defect analysis (predict where bugs are likely to occur).
- Automated regression testing and continuous testing with minimal human effort.
- 5. Deployment
- AI-driven DevOps pipelines with self-optimizing deployments.
- Predicting downtime, performance issues, and rollback needs.
- Intelligent container orchestration (AI + Kubernetes).

- 6. Maintenance & Monitoring
- Predictive maintenance using AI (detecting failures before they occur).
- AI chatbots for user issue resolution.
- Automated software updates and adaptive feature improvements.
- 7. Security Enhancements
- AI-powered threat detection and vulnerability scanning.
- Behavioral analysis of software to detect abnormal activities.
- Smart encryption and self-defensive mechanisms.
- 8. Project Management & Collaboration
- AI-powered task allocation to optimize productivity.
- Intelligent monitoring of developer performance.
- Virtual project managers powered by AI.