SMARTSDLC – AL – Enhanced Software Development Lifecycle Documentation

1. INTRODUCTION

- **Project Title: SMARTSDLC – AL – Enhanced Software Development Lifecycle**
- **Team Members:**
    - Team Member 1: Divya Priya V
    - Team Member 2: Monisha V
    - Team Member 3: Hema Harini
    - Team Member 4: Almas A
    - Team Member 5: Prathima

2. PROJECT OVERVIEW

Purpose
SmartSDLC is an AI-powered platform designed to automate and streamline the Software Development Lifecycle (SDLC). It leverages IBM Watsonx Granite models, LangChain, FastAPI, and Streamlit to enhance phases of software engineering, including requirements analysis, code generation, test case creation, bug fixing, and documentation. By integrating AI-driven automation, SmartSDLC reduces manual workload, accelerates development timelines, and improves software quality.

FEATURES

- **Requirement Upload & Classification**
    - ***Key Point:*** Structured requirement management
    - ***Functionality:*** Extracts text from uploaded PDFs and classifies sentences into SDLC phases (Requirements, Design, Development, Testing, Deployment).
- **AI Code Generator**
    - ***Key Point:*** Automated code creation
    - ***Functionality****:* Generates clean, production-ready code from natural language prompts or structured user stories.
- **Bug Fixer**
    - ***Key Point:*** Error detection and correction
    - ***Functionality:*** Identifies and fixes syntax and logic errors in code snippets, returning optimized versions**.**
- **AI-Driven Test Case Generation**
    - ***Key Point:*** Automated testing support

- ***Functionality**:* Generates unit and integration test cases from generated or user-provided code.

- **Code Summarization & Documentation**
  - ***Key Point**:* Improved maintainability
  - ***Functionality:*** Summarizes and documents code for better readability and project understanding**.**
- **Chatbot Assistance**
  - ***Key Point**:* Real-time developer support
  - ***Functionality**:* Provides interactive guidance and assistance for SDLC-related queries.
- **GitHub Integration**
  - ***Key Point:*** Workflow automation
  - ***Functionality:*** Automates pushing code, opening issues, and syncing documentation with GitHub repositories**.**

USE CASE SCENARIOS

- Upload raw requirement PDFs and receive structured user stories grouped by SDLC phase.
- Generate working Python or JavaScript code from natural language prompts.
- Submit buggy code and receive AI-optimized corrections.
- Automatically generate test cases for faster validation.
- Access an AI-powered assistant for SDLC queries and guidance**.**

3. ARCHITECTURE

- Frontend (Streamlit): Interactive dashboard for requirements, code generation, bug fixing, testing, and chatbot interaction.
- Backend (FastAPI): API layer handling routing, authentication, AI requests, and service orchestration.
- AI Integration (IBM Watsonx + LangChain): Natural language processing, code generation, bug fixing, and summarization.
- Modules: Requirement analysis, code generation, bug fixing, test case generation, code summarization, GitHub workflows.
- Deployment: Local hosting with Uvicorn (backend) and Streamlit (frontend).

4. SETUP INSTRUCTIONS

PREREQUISITES

- Python 3.10+
- FastAPI, Uvicorn
- Streamlit
- IBM Watsonx API access
- LangChain
- PyMuPDF (fitz)
- Git & GitHub

INSTALLATION PROCESS

1. Install Python 3.10 and pip.
2. Create virtual environment:
3. python -m venv myenv
4. Activate environment and install dependencies from requirements.txt.
5. Configure .env file with API keys and model IDs.
6. Start FastAPI backend:
7. uvicorn app.main:app --reload
8. Run Streamlit frontend:
9. streamlit run frontend/Home.py

5. FOLDER STRUCTURE

- **app/ –** FastAPI backend
    - routes/ – API endpoints for AI, chat, auth, feedback
    - services/ – Core AI service logic
    - models/, utils/ – Supporting modules
- **frontend/ –** Streamlit UI components
    - Home.py – Entry dashboard
    - pages/ – Modular pages (requirements, code generator, bug fixer, etc.)
- **ai_story_generator.py** – Requirement classification logic
- **code_generator.py** – Code and test case generation
- **bug_resolver.py** – Bug fixing automation

- **doc_generator.py –** Code summarization
- **conversation_handler.py –** Chatbot logic
- **github_service.py –** GitHub workflow automation

6. RUNNING THE APPLICATION

1. Start FastAPI backend with Uvicorn.
2. Run Streamlit dashboard.
3. Navigate using the dashboard menu.
4. Upload requirements or enter prompts.
5. Generate code, fix bugs, create tests, and access chatbot.
6. Sync outputs with GitHub and export documentation**.**

7. API DOCUMENTATION

- POST /upload-pdf – Uploads requirements for classification
- POST /generate-code – Generates production-ready code
- POST /fix-bugs – Accepts buggy code and returns corrected version
- POST /generate-tests – Creates test cases
- POST /summarize-code – Summarizes uploaded code
- POST /chat – Chatbot interactions
- POST /feedback – Submits user feedback
- GET /docs – Swagger UI for API exploration

8. AUTHENTICATION

- Token-based authentication (JWT)
- Role-based access (admin, developer, tester)
- Hashed user login and registration
- Planned: OAuth2 integration and session management

9. USER INTERFACE

- Home Dashboard: Feature overview and navigation

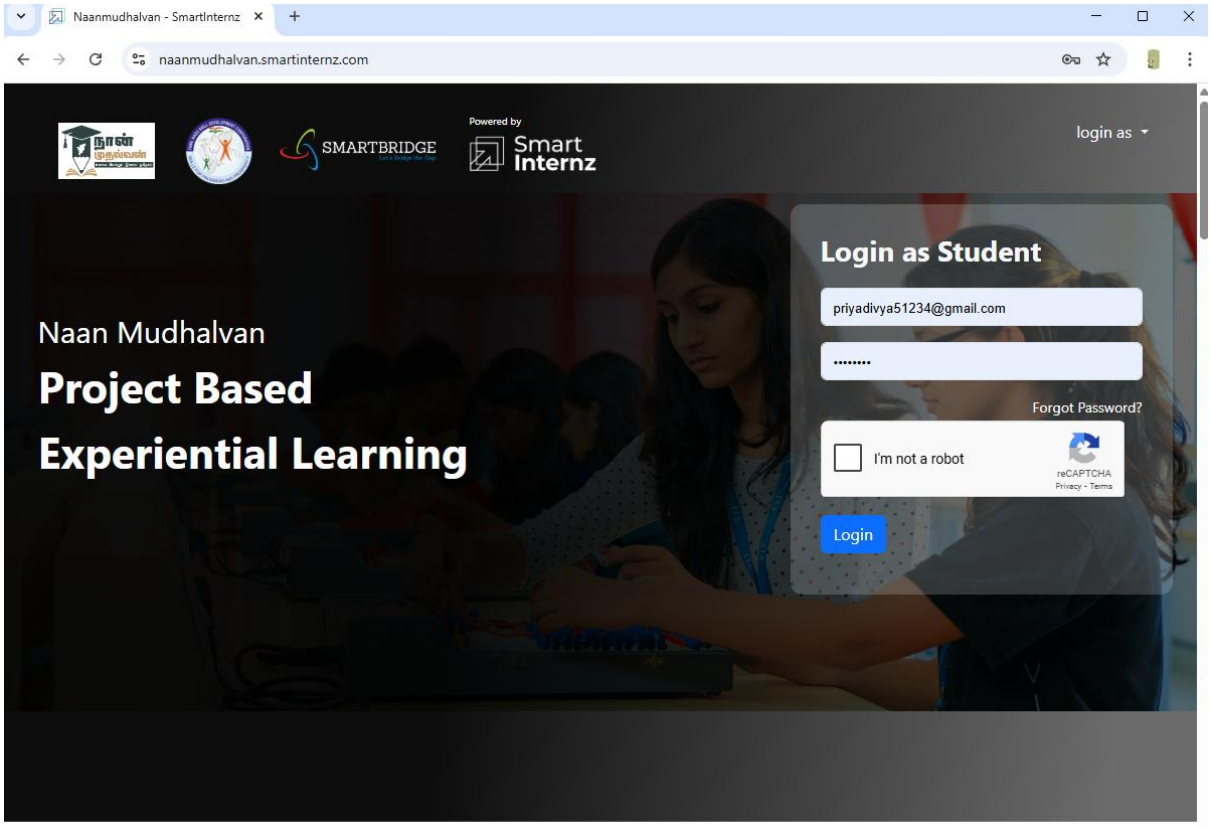
- Requirement Classifier: Upload and classify requirements
- Code Generator: Prompt-based code creation
- Bug Fixer: Code correction interface
- Test Generator: Auto-generated test cases
- Chatbot: Real-time AI guidance
- Feedback Form: Collects user feedback
- GitHub Sync: Push code, open issues, sync docs

10. TESTING

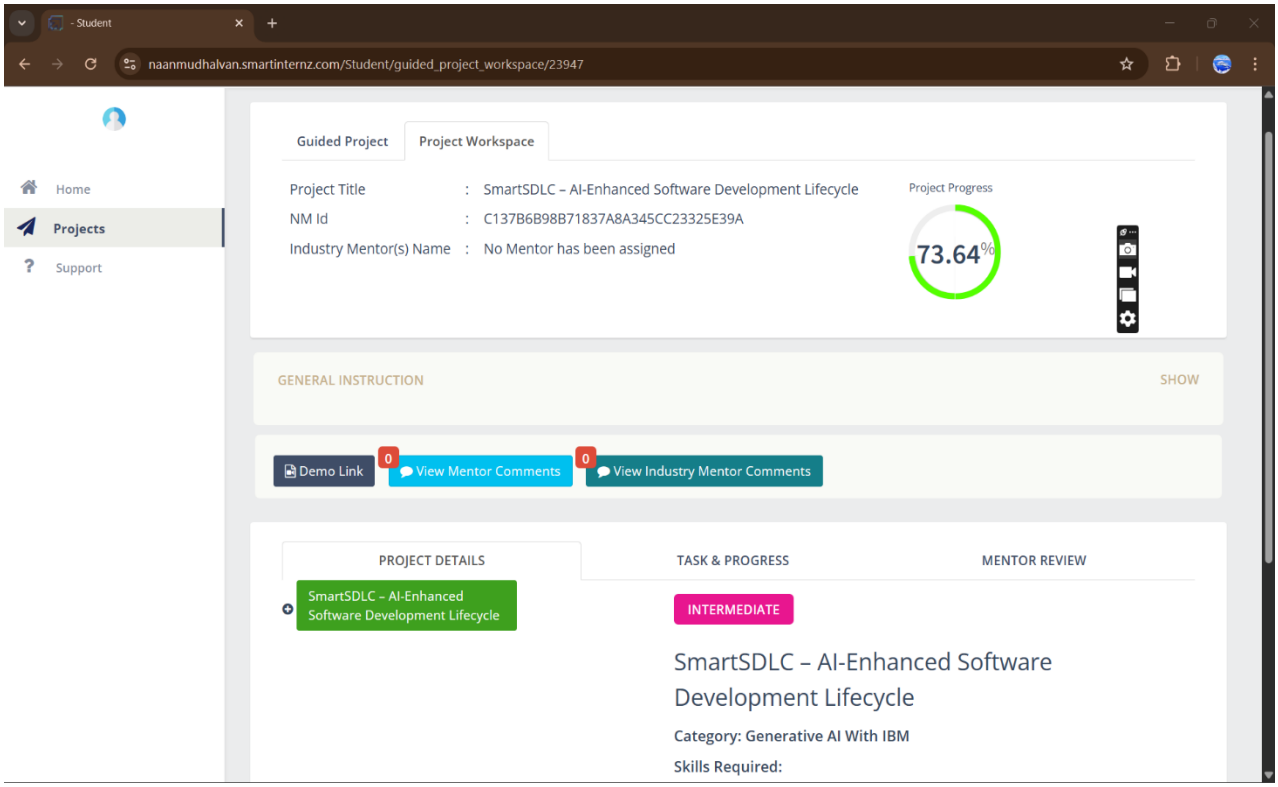
- Unit Testing: For backend AI services
- API Testing: Swagger UI and Postman
- Manual Testing: Requirement classification, bug fixing, chatbot functionality
- Edge Cases: Large PDF uploads, malformed prompts, incorrect API keys

11. SCREENSHOTS

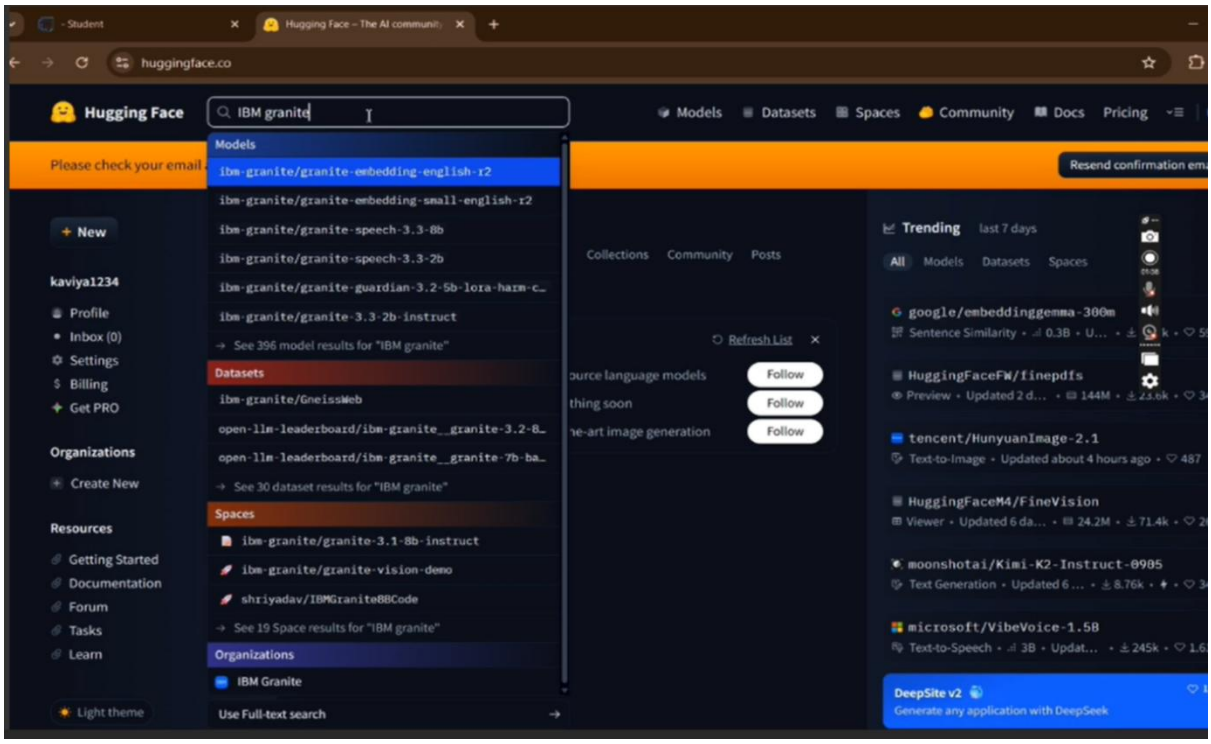
STEPS TO IMPLEMENT :



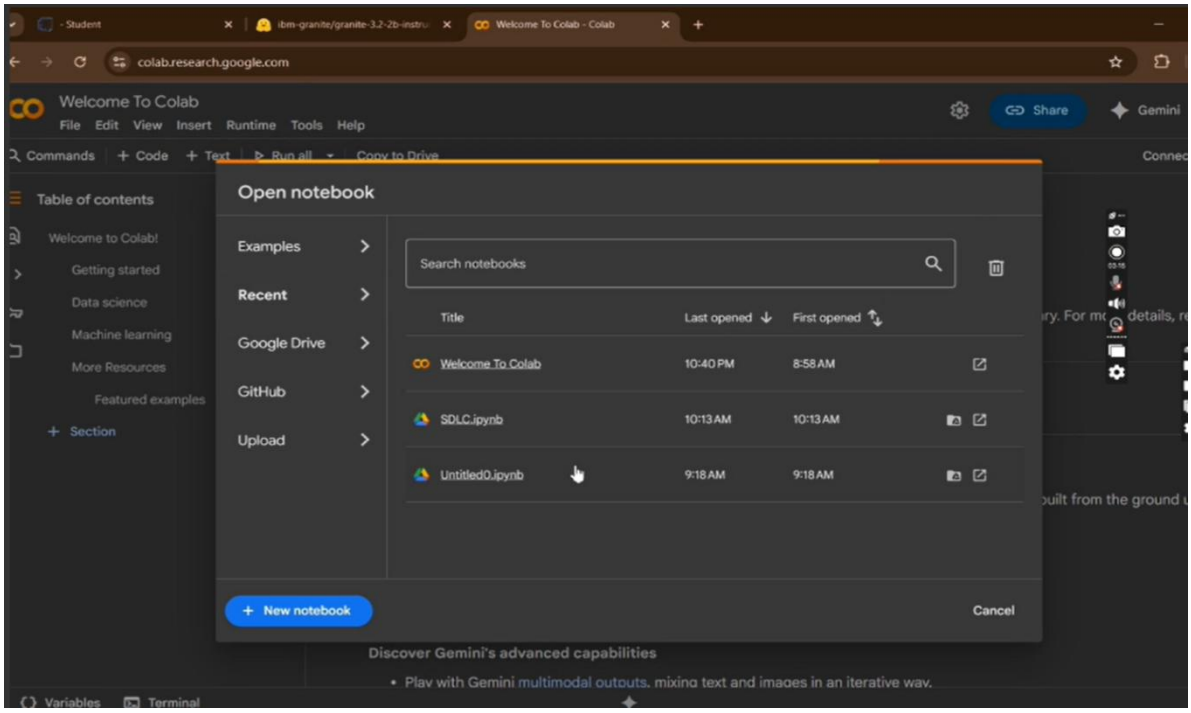
SMARTINTERNZ LOGIN PAGE!



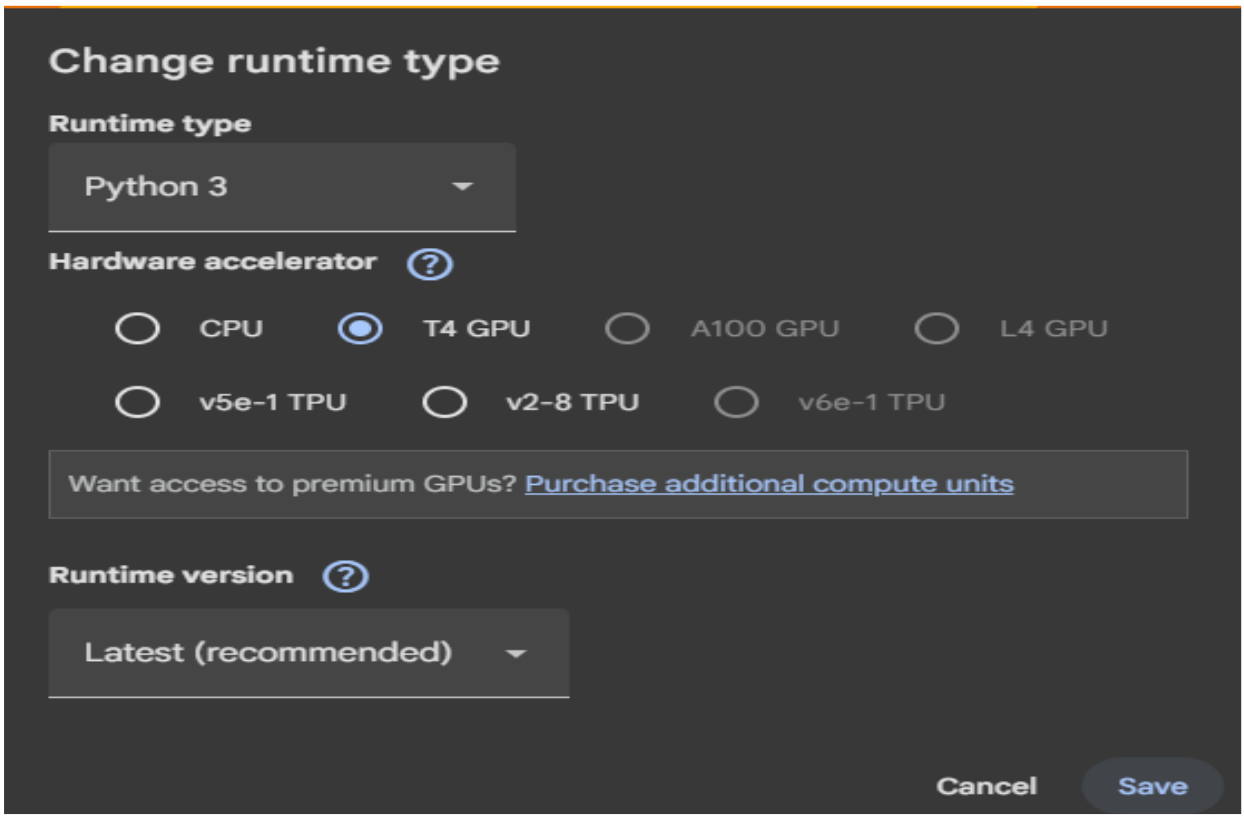
SMARTINTERNZ DASHBOARD



granite-3.2-2b-instruct in hugging face



NEW NOTEBOOK -NOTEBOOK PAGE IN GOOGLE COLAB



CHOOSE "T4 GPU" AND CLICK ON "SAVE"

```python
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
```

```python
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):
    # Get text from PDF or direct input
    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 important points, key provisions, and implications:\n\n{content}"
    else:
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{policy_text}"

    return generate_response(summary_prompt, max_length=1200)

# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# Eco Assistant & Policy Analyzer")

    with gr.Tabs():
        with gr.TabItem("Eco Tips Generator"):
            with gr.Row():
                with gr.Column():
                    keywords_input = gr.Textbox(
                        label="Environmental Problem/Keywords",
```

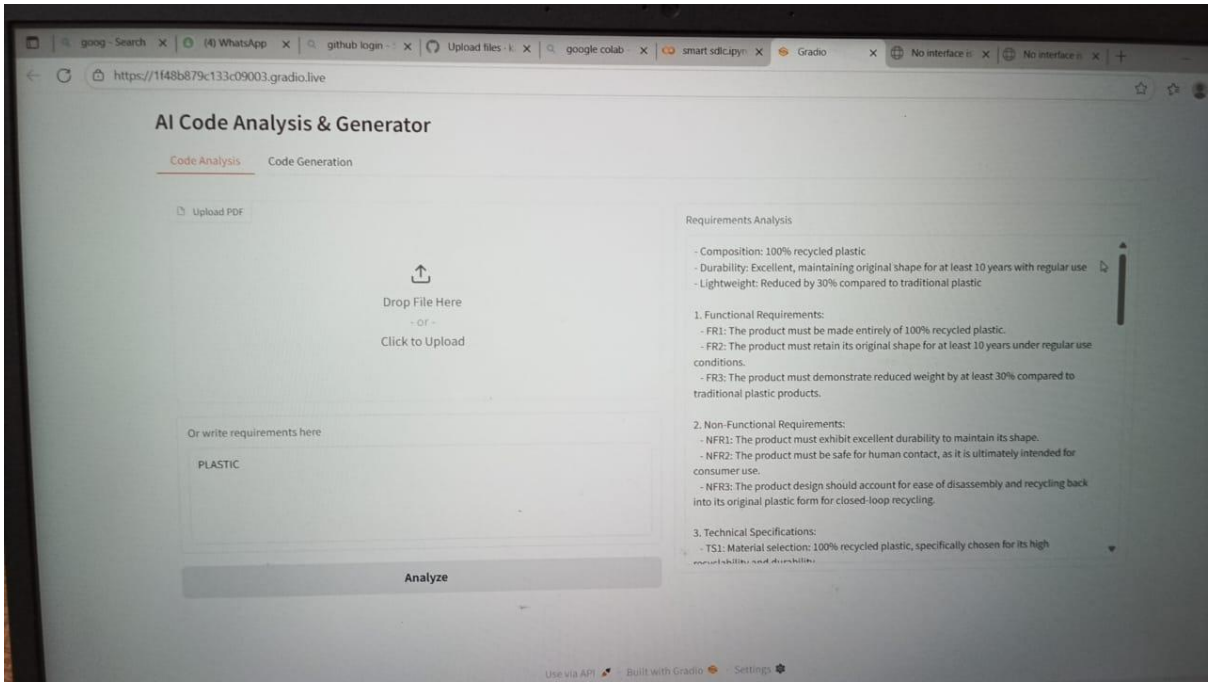
```python
# Create Gradio Interface
with gr.Blocks() as app:
    gr.Markdown("# Eco Assistant & Policy Analyzer")

    with gr.Tabs():
        with gr.TabItem("Eco Tips Generator"):
            with gr.Row():
                with gr.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")

                with gr.Column():
                    tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)

            generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)

        with gr.TabItem("Policy Summarization"):
            with gr.Row():
                with gr.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")

                with gr.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)
```

SMART SDLC CODING PAGE!

OUTPUT :

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your sessi
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 8.88k/? [00:00<00:00, 274kB/s]
vocab.json: 777k/? [00:00<00:00, 6.96MB/s]
merges.txt: 442k/? [00:00<00:00, 8.61MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 36.3MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 2.03kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 14.2kB/s]
config.json: 100% 786/786 [00:00<00:00, 21.6kB/s]
model.safetensors.index.json: 29.8k/? [00:00<00:00, 1.00MB/s]
Fetching 2 files: 100% 2/2 [01:17<00:00, 77.09s/it]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:10<00:00, 6.03MB/s]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:16<00:00, 62.0MB/s]
Loading checkpoint shards: 100% 2/2 [00:19<00:00, 8.04s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 7.02kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://5475fe6c096b7ff650.gradio.live
```

CLICK ON THE URI TO OPEN THE GRADIO APPLICATION CLICK ON THE LINK!



CODE ANALYSIS PAGE

12.KNOWN ISSUES

- Limited offline support
- Dependency on IBM Watsonx API availability
- Occasional latency for large PDFs
- Basic test case generation (needs extension)

13. FUTURE ASSESMENTS

- CI/CD pipeline integration
- Multi-language support for code generation
- Advanced bug detection with deep learning
- Cloud deployment (AWS, IBM Cloud, Azure)
- Collaboration features for team workflows
- Enhanced test generation with coverage analysis