

SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation

1.Introduction

Project title :

SmartSDLC – AI-Enhanced Software Development Lifecycle

Team members:

1. Rakshitha K (Team leader)
2. Jeborah Magdalin Doss M
3. Gayathri E
4. Vinotha E
5. Janani R G

2. Project Overview

Purpose:

The purpose of SmartSDLC is to streamline and accelerate the software development lifecycle by leveraging the Granite model from Hugging Face. By integrating AI capabilities into each phase—requirements gathering, coding, testing, debugging, documentation, and interactive support—SmartSDLC empowers developers to move from ideas to working solutions more efficiently. Deployed in Google Colab for easy accessibility and reliable performance, the project aims to simplify complex workflows, reduce development time, and enhance productivity for students, professionals, and teams alike.

Features:

1. **Requirement Extraction from PDFs**
 - Upload software-related documents in PDF format.
 - Automatically extracts and organizes requirements into **functional**, **non-functional**, and **technical** categories.
2. **Prompt-to-Code Generation**
 - Converts natural language requirements into working code.
 - Supports multiple programming languages (Python, Java, C++, JavaScript, etc.).
3. **Automated Test Case Generation**
 - Generates **unit tests** and **integration tests** from requirements or code.
 - Ensures correctness and reduces manual test writing.
4. **Bug Detection & Fixing**
 - Analyzes user-provided code.
 - Identifies logical or syntax errors and provides corrected code with explanations.
5. **Documentation Generator**
 - Creates clean and structured documentation from code.
 - Explains functions, classes, and workflows for easy understanding.
6. **Interactive AI Chat Assistant**
 - AI-powered helper for queries on coding, debugging, and software development practices.
 - Provides real-time support like a virtual mentor.
7. **Google Colab Deployment**
 - Runs directly in Google Colab with no setup hassle.
 - Accessible from anywhere with consistent performance.

3. Architecture

- **Input Layer:** Accepts PDFs, text prompts, code, or queries from users.
- **Processing Layer:** Extracts text, tokenizes inputs, and passes them to the Granite AI model.
- **AI Core Engine:** Granite model (via LangChain) processes tasks such as requirement analysis, code generation, and summarization.
- **Functional Modules:**
 - Requirement Analyzer
 - Code Generator
 - Test Case Generator
 - Bug Fixer
 - Documentation Assistant
 - Chatbot Helper
- **Interface Layer:** User interaction via Streamlit, FastAPI, or Google Colab.
- **Deployment Layer:** Cloud-hosted environment with GPU/CPU acceleration.

Workflow:

User Input → Preprocessing → Granite AI Engine → Functional Module → Output to UI

4. Setup Instructions

Prerequisites:

1. **Python:** Version 3.10+
[Download Python 3.10](#)
2. **PyTorch:** Compatible with your system (CPU or GPU)
PyTorch Installation Guide

3. **Transformers library** (Hugging Face)
For loading IBM Granite model: transformers>=4.30.0
4. **Gradio**
For building the UI: gradio>=3.50
5. **PyPDF2**
For extracting text from PDF files: PyPDF2>=3.0.0
6. **Optional (GPU)**
 - CUDA toolkit installed (for faster inference)
 - NVIDIA GPU drivers

Installation:

1. **Clone/ Download the Project**
 - `git clone <repository_link>`
`cd SmartSDLC`
 2. **Install Dependencies**
 - `pip install torch transformers gradio PyPDF2`
 3. **Run the Application**
 - `python app.py`
 4. **In Google Colab (Recommended)**
 - Upload project files to Colab.
 - Install dependencies inside Colab:
 - `!pip install torch transformers gradio PyPDF2`
 5. **Run the main script:**
 - `app.launch(share=True)`
 6. **Access the Interface**
 - A public URL will be generated via Gradio.
 - Use the tabs to:
 - Upload PDFs & extract requirements
 - Generate code
 - Create tests
 - Debug
 - Chat with the assistant
-

5. Folder Structure

- SmartSDLC # Root Project folder
 - app.py # Main Gradio application script
 - requirements.txt # Optional: List of dependencies
 - README.md # Project documentation
 - utils/ # Helper functions (optional)
 - pdf_utils.py # PDF extraction functions
 - model_utils.py # Model inference functions
 - notebooks # Optional: Jupyter notebooks
 - data # Sample PDFs or input files
 - outputs # Generated code or analysis results
-

6. Running the Application

Locally

1. Open terminal / command prompt.
2. Navigate to the project folder:
 - `cd SmartSDLC`
3. Install dependencies (if not done already):
 - `pip install torch transformers gradio PyPDF2`
4. Run the main app:
 - `python app.py`
5. A local URL will appear (e.g., `http://127.0.0.1:7860`) in the terminal. Open it in your browser to use the interface.

In Google Colab

1. Upload project files to Colab.
 2. Install required libraries inside a notebook:
 - `!pip install torch transformers gradio PyPDF2`
 3. Run the app in a cell:
 - `app.launch(share=True)`
 4. Gradio will generate a public URL to access the interface from any browser.
-

7. API Documentation

1. **generate_response (prompt, max_length=1024)**
 - Generates a response using the IBM Granite model.
 - **Input:** Text prompt (string) and optional max length (default 1024).
 - **Output:** Generated text (string).
2. **extract_text_from_pdf (pdf_file)**
 - Extracts text from an uploaded PDF file.
 - **Input:** PDF file.
 - **Output:** Text from the PDF, or error message if extraction fails.
3. **requirement_analysis (pdf_file, prompt_text)**
 - Analyzes requirements from PDF or text and organizes them into functional, non-functional, and technical requirements.
 - **Input:** PDF file (optional) or requirement text.
 - **Output:** Requirement analysis as text.
4. **code_generation (prompt, language)**
 - Generates code based on the requirement in the selected programming language.

- **Input:** Requirement text and programming language (Python, JavaScript, Java, C++, C#, PHP, Go, Rust).
- **Output:** Generated code as text.

5. Gradio Interface

Code Analysis Tab:

- Upload PDF or enter text.
- Click “Analyze” to see organized requirements.

Code Generation Tab:

- Enter requirement text and select programming language.
 - Click “Generate Code” to see the generated code.
-

8. Authentication

Current Status:

- The app currently has **no authentication**; anyone with access to the URL can use it.

Future Plans:

- Implement **user login** (email/password or OAuth).
 - Add **API key protection** for secure access.
 - Optionally enable **role-based access** (e.g., admin vs user).
-

9. User Interface

The SmartSDLC app has a clean, tabbed interface built with Gradio.

1. Code Analysis Tab

- Upload a PDF or enter requirements in the textbox.
- Click “Analyze” to get functional, non-functional, and technical requirements.
- Results appear in a textbox below.

2. Code Generation Tab

- Enter a requirement in the textbox and select a programming language from the dropdown.
- Click “Generate Code” to get the generated code in the textbox.

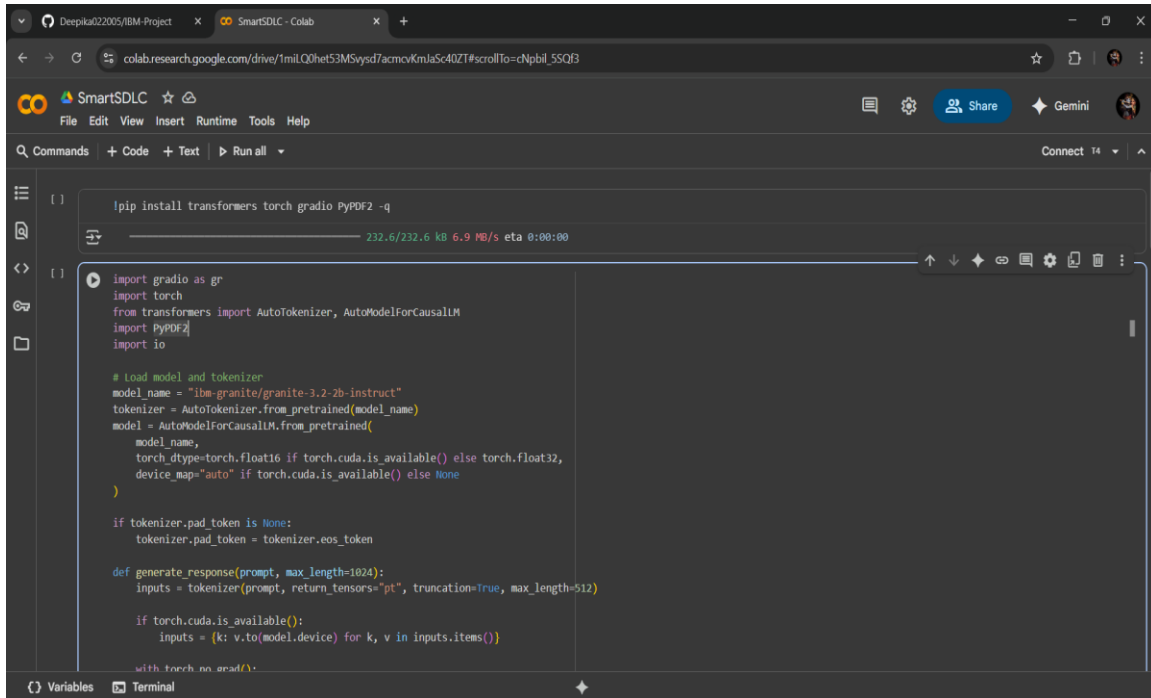
3. Features

- Simple and easy-to-use layout.
 - Tabs separate analysis and code generation for clarity.
 - Works on desktop and mobile browsers.
-

10. Testing

- Upload sample PDFs to test **requirement extraction**.
 - Enter requirement text directly to test **text-based analysis**.
 - Generate code in different programming languages to verify **code generation**.
 - Check that the **Gradio interface** works correctly, including buttons, textboxes, and dropdowns.
 - Ensure the output is accurate and properly formatted for both analysis and code generation.
-

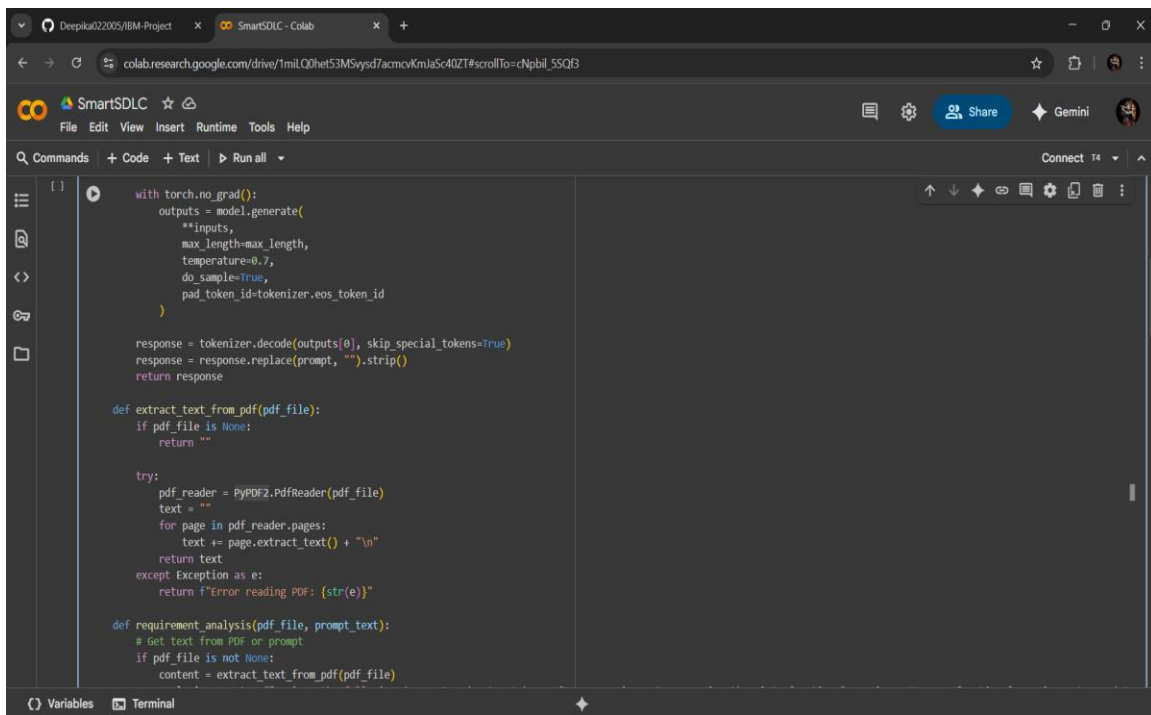
11. Source Code



The screenshot shows a Google Colab notebook interface. The top bar includes the SmartSDLC logo, a star icon, and a share button. The menu bar contains File, Edit, View, Insert, Runtime, Tools, and Help. The left sidebar has icons for file explorer, code editor, and terminal. The main code editor area contains the following Python code:

```
[ ]  
!pip install transformers torch gradio PyPDF2 -q  
  
[ ]  
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():
```

The bottom of the interface shows tabs for Variables and Terminal.



The screenshot shows the same Google Colab notebook interface. The code editor area contains the following Python code:

```
[ ]  
    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  
  
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 requirement_analysis(pdf_file, prompt_text):  
    # Get text from PDF or prompt  
    if pdf_file is not None:  
        content = extract_text_from_pdf(pdf_file)
```

The bottom of the interface shows tabs for Variables and Terminal.

```
SmartSDLC - Colab
colab.research.google.com/drive/1mILQ0het53MSysd7acmcvKmlaSc40ZT#scrollTo=cNpbil_5SQJ3

SmartSDLC
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all
Connect T4

analysis_prompt = f"Analyze the following document and extract key software requirements. Organize them into functional requirement:
else:
    analysis_prompt = f"Analyze the following requirements and organize them into functional requirements, non-functional requirements, and technical specifications:\n\n[pro

return generate_response(analysis_prompt, max_length=1200)

def code_generation(prompt, language):
    code_prompt = f"Generate {language} code for the following requirement:\n\n(prompt)\n\ncode:"
    return generate_response(code_prompt, max_length=1200)

# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# AI Code Analysis & Generator")

    with gr.Tabs():
        with gr.Tabitem("Code Analysis"):
            with gr.Row():
                with gr.Column():
                    pdf_upload = gr.File(label="Upload PDF", file_types=[".pdf"])
                    prompt_input = gr.Textbox(
                        label="Or write requirements here",
                        placeholder="Describe your software requirements...",
                        lines=5
                    )
                    analyze_btn = gr.Button("Analyze")

                with gr.Column():
                    analysis_output = gr.Textbox(label="Requirements Analysis", lines=20)

            analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input], outputs=analysis_output)
```

```
SmartSDLC - Colab
colab.research.google.com/drive/1mILQ0het53MSysd7acmcvKmlaSc40ZT#scrollTo=cNpbil_5SQJ3

SmartSDLC
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all
Connect T4

analyze_btn = gr.Button("Analyze")

with gr.Column():
    analysis_output = gr.Textbox(label="Requirements Analysis", lines=20)

analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input], outputs=analysis_output)

with gr.Tabitem("Code Generation"):
    with gr.Row():
        with gr.Column():
            code_prompt = gr.Textbox(
                label="Code Requirements",
                placeholder="Describe what code you want to generate...",
                lines=5
            )
            language_dropdown = gr.Dropdown(
                choices=["Python", "JavaScript", "Java", "C++", "C#", "PHP", "Go", "Rust"],
                label="Programming Language",
                value="Python"
            )
            generate_btn = gr.Button("Generate Code")

        with gr.Column():
            code_output = gr.Textbox(label="Generated Code", lines=20)


    generate_btn.click(code_generation, inputs=[code_prompt, language_dropdown], outputs=code_output)

app.launch(share=True)

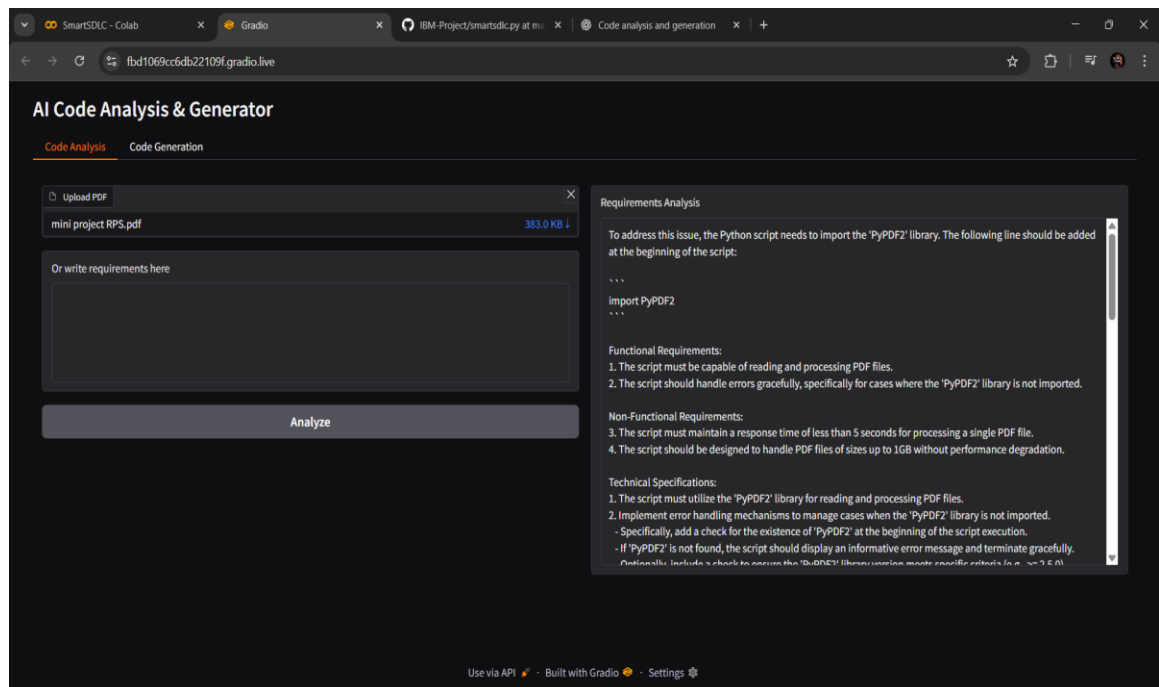
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
```

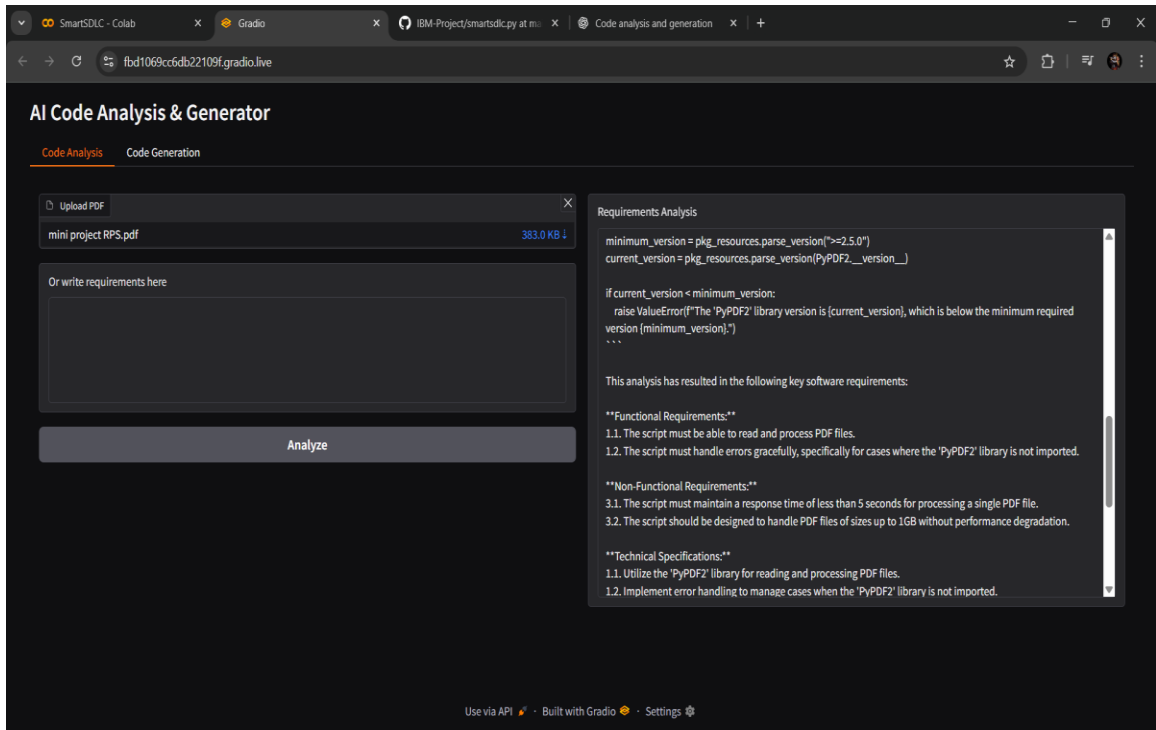
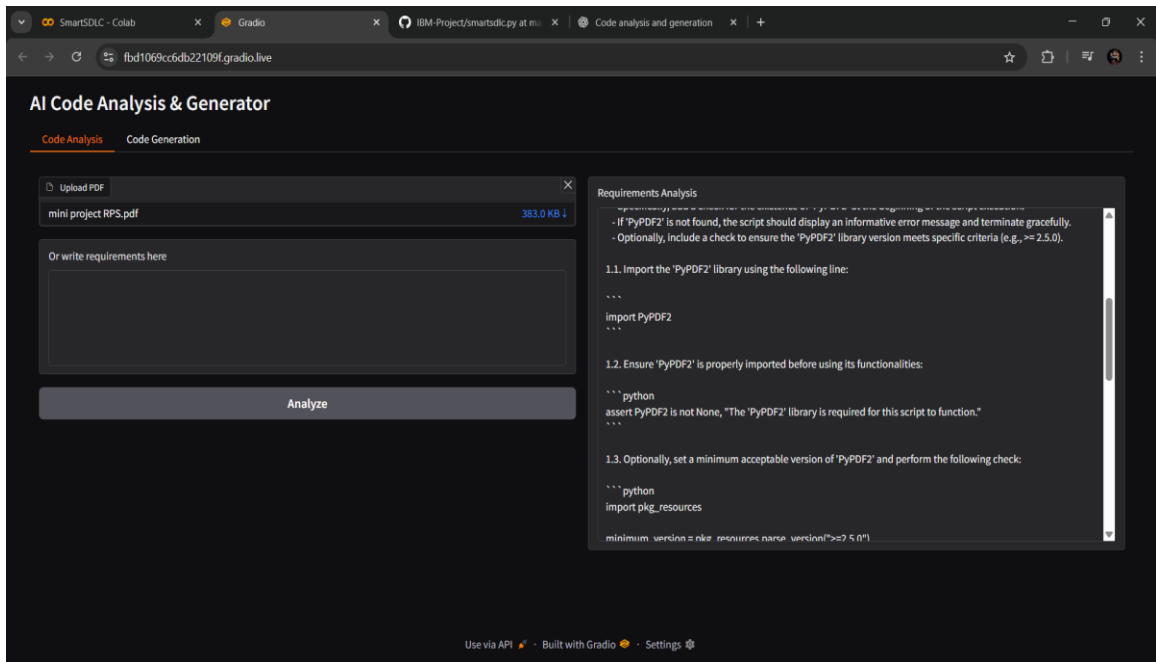
12. Output

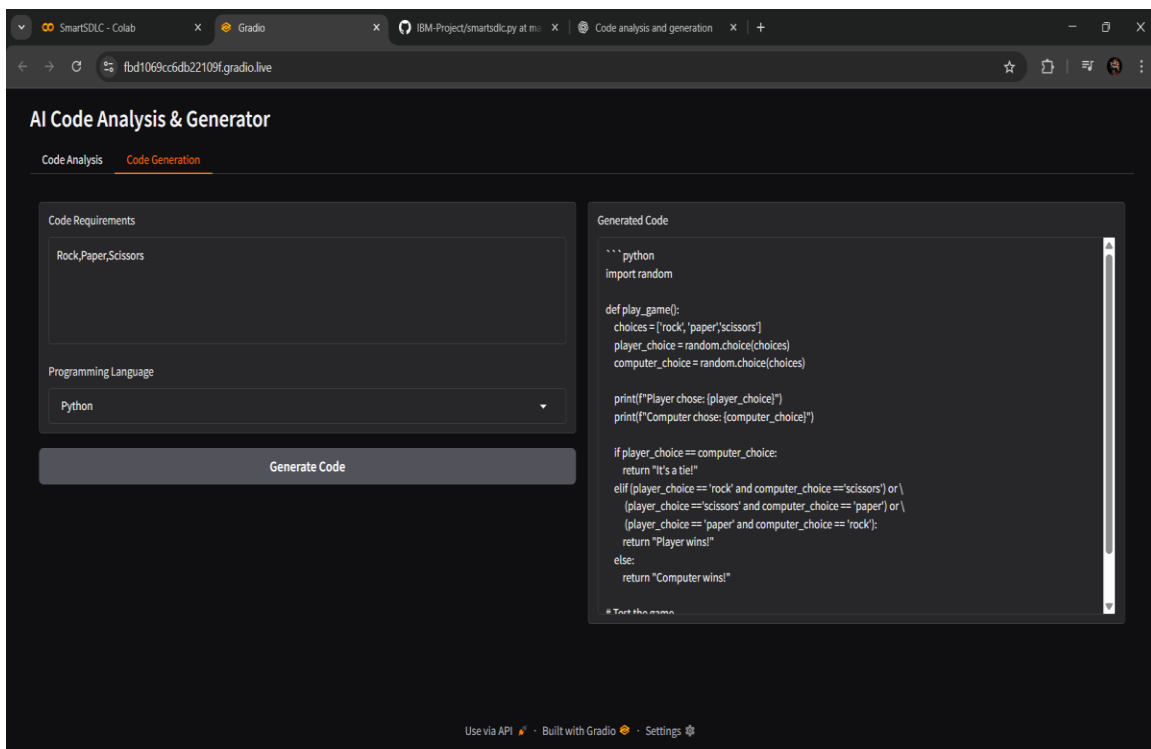
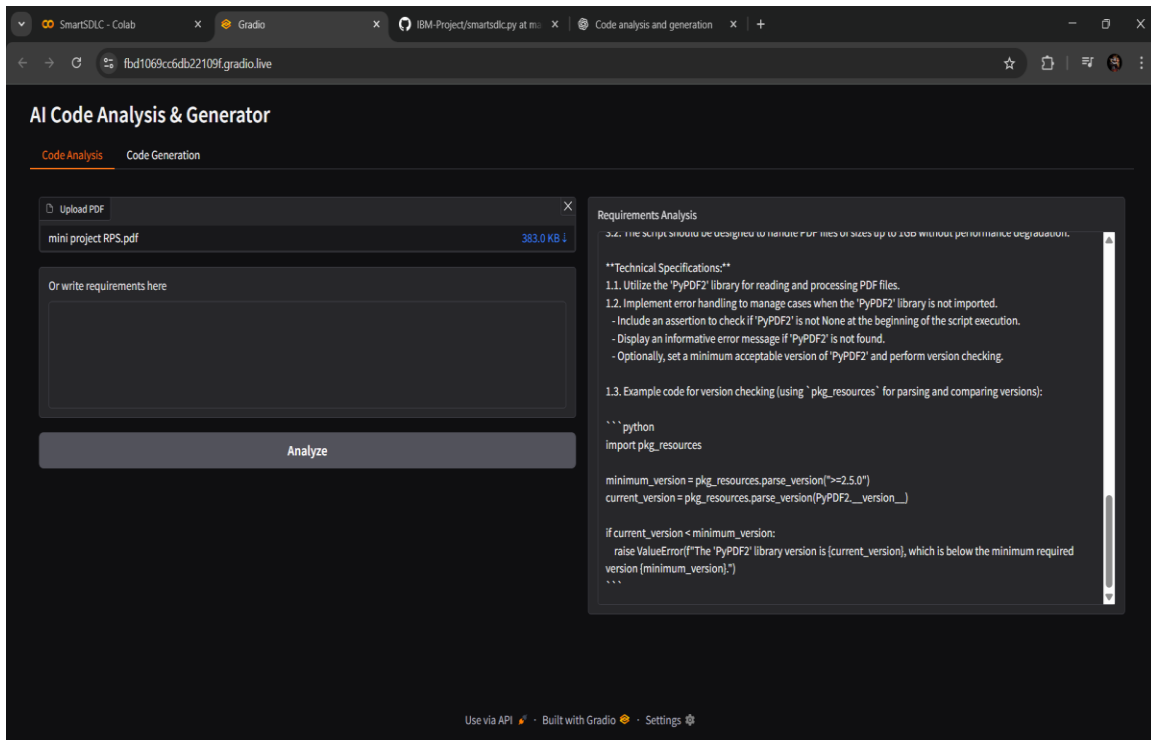
```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret 'HF_TOKEN' does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
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, 750kB/s]
vocab.json 777k? [00:00<00:00, 1.97MB/s]
merges.txt 442k? [00:00<00:00, 23.5MB/s]
tokenizer.json 3.48M? [00:00<00:00, 91.3MB/s]
added_tokens.json 100% [00:00<00:00, 10.0kB/s]
special_tokens_map.json 100% [00:00<00:00, 77.7kB/s]
config.json 100% [00:00<00:00, 63.0kB/s]
'torch_dtype' is deprecated! Use 'dtype' instead!
model.safetensors.index.json 29.8k? [00:00<00:00, 2.40MB/s]
Fetching 2 files: 100% [03:44<00:00, 224.76s/it]
model-00002-of-00002.safetensors: 100% [00:01<00:00, 31.2MB/s]
model-00001-of-00002.safetensors: 100% [03:44<00:00, 46.2MB/s]
Loading checkpoint shards: 100% [00:21<00:00, 9.05s/it]
generation_config.json 100% [00:00<00:00, 9.44kB/s]
colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://a23b6c089e95c4fb98.gradio.live
this share link expires in 1 week. For free permanent hosting and GPU upgrades, run 'gradio deploy' from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)

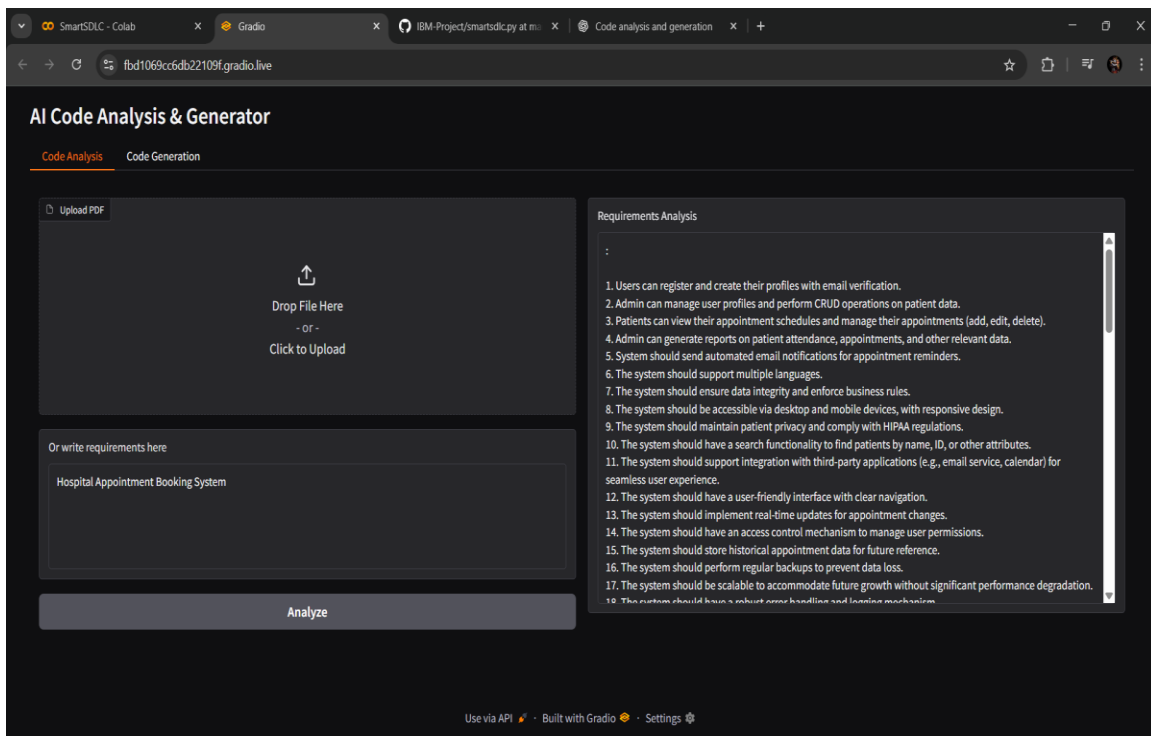
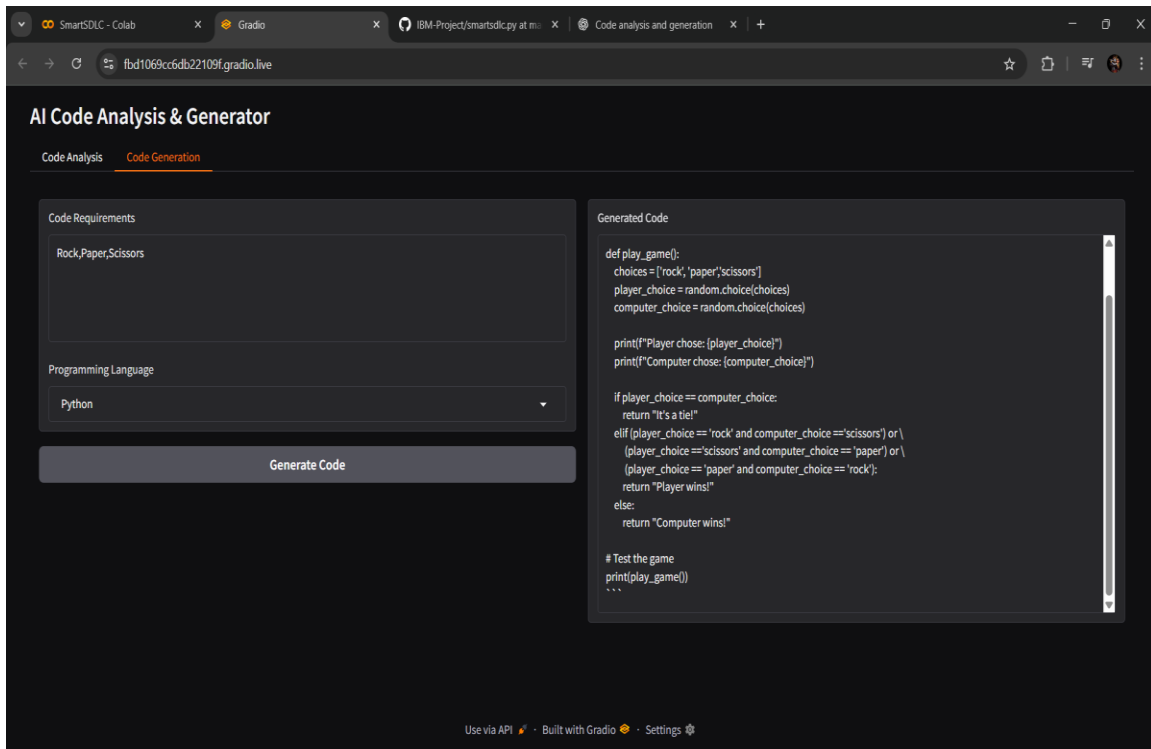


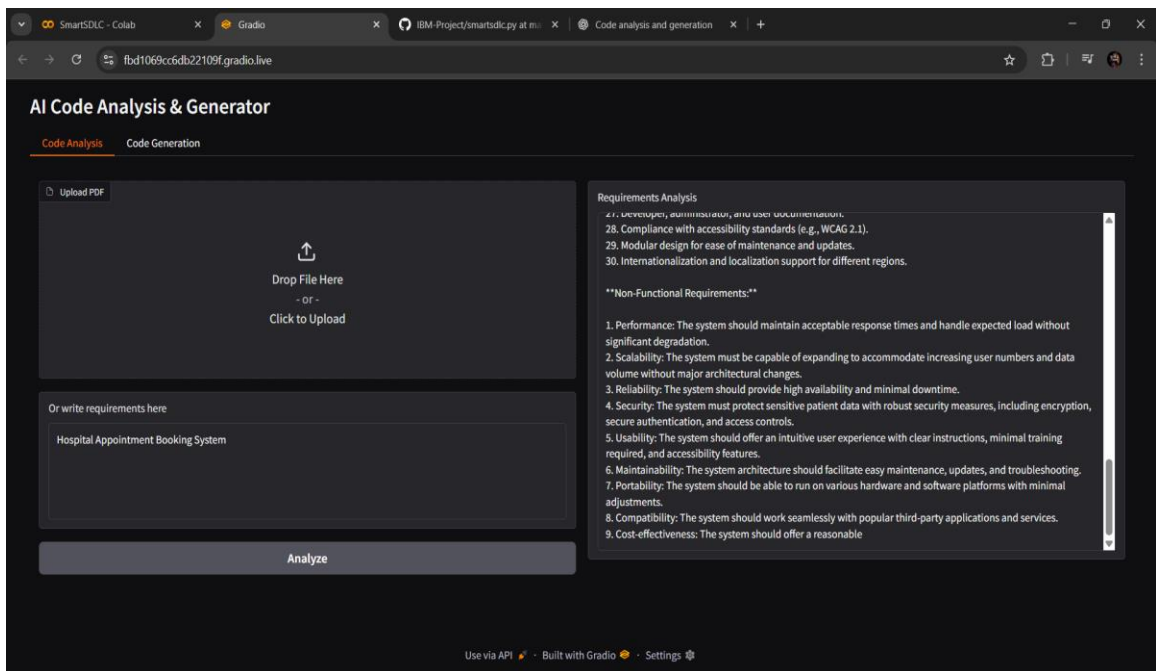
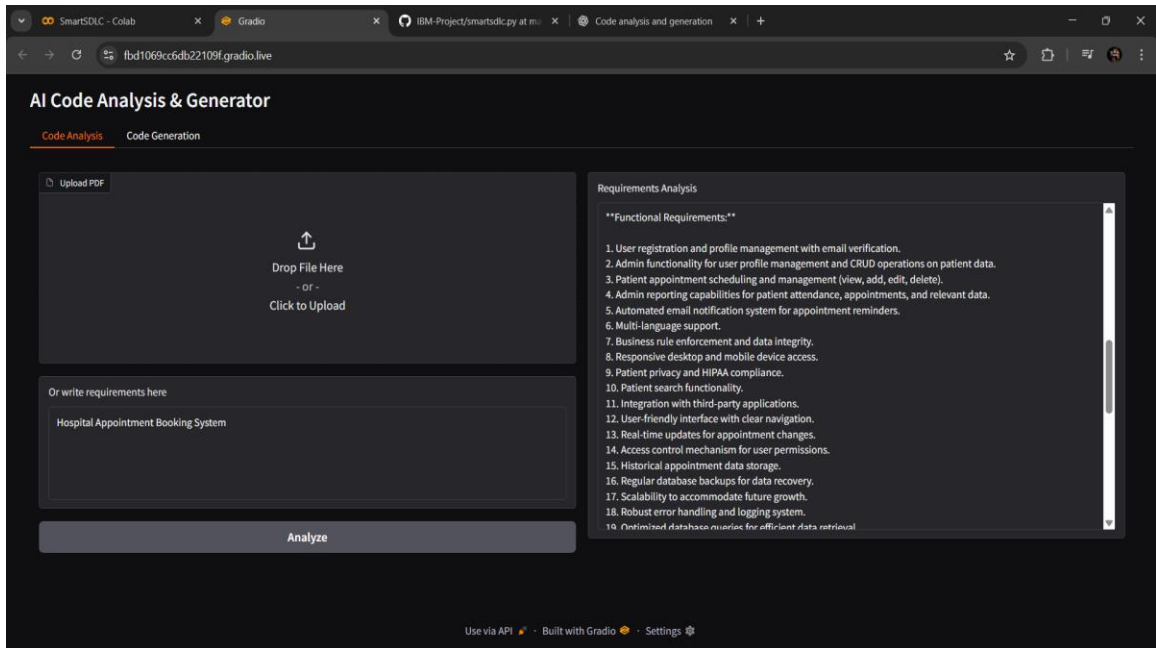
No interface is running right now
```

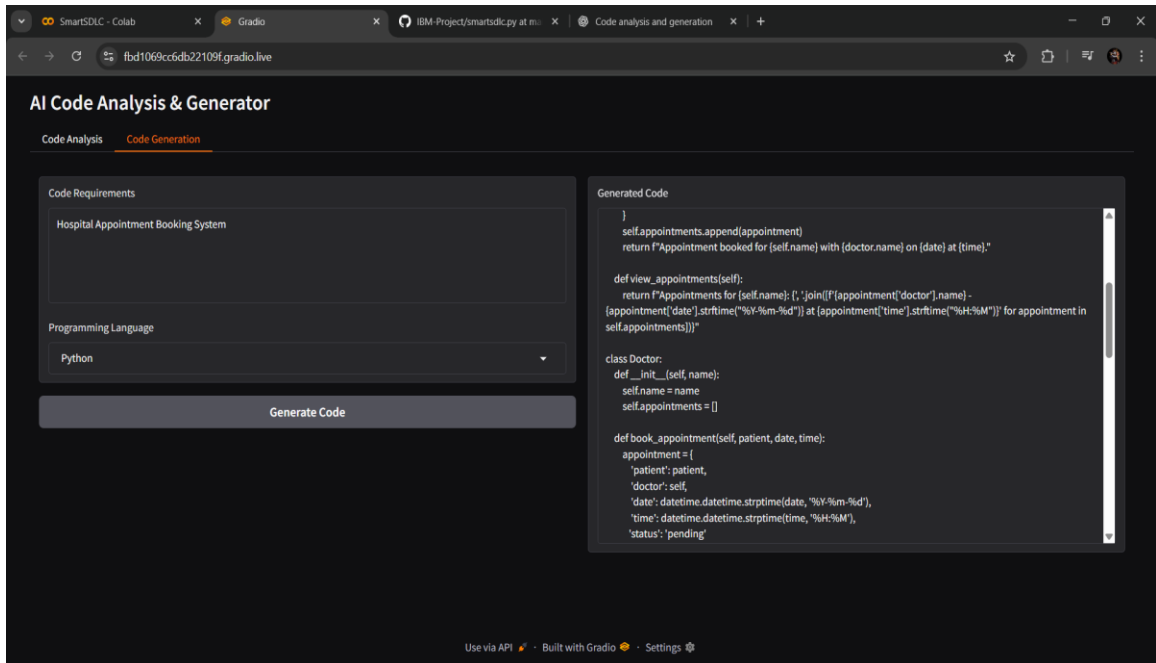
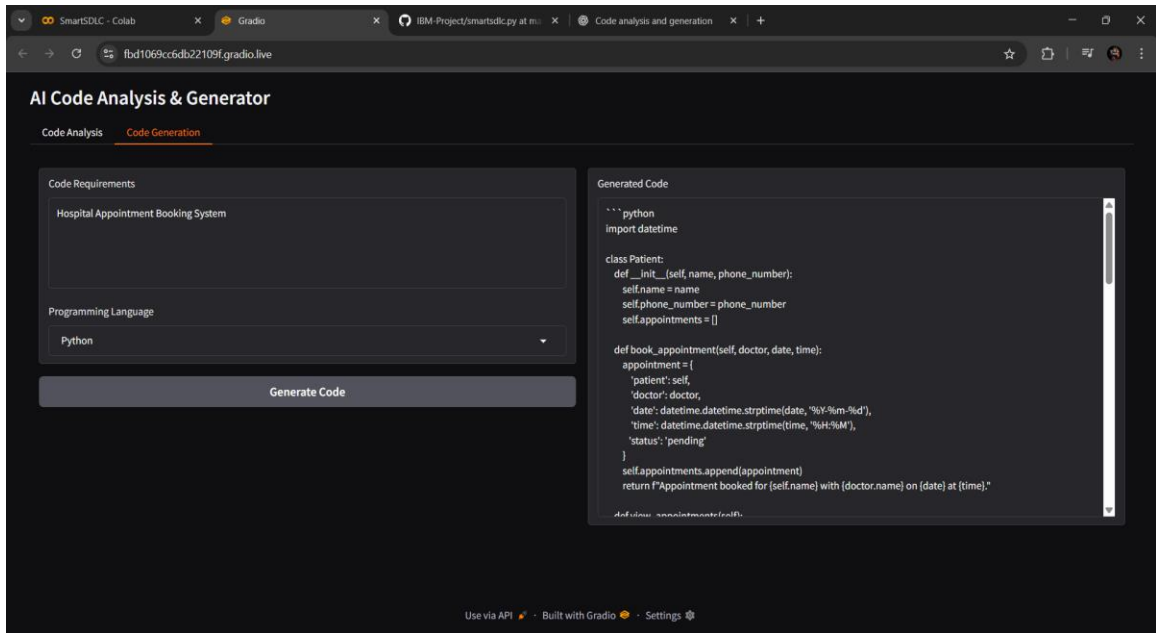


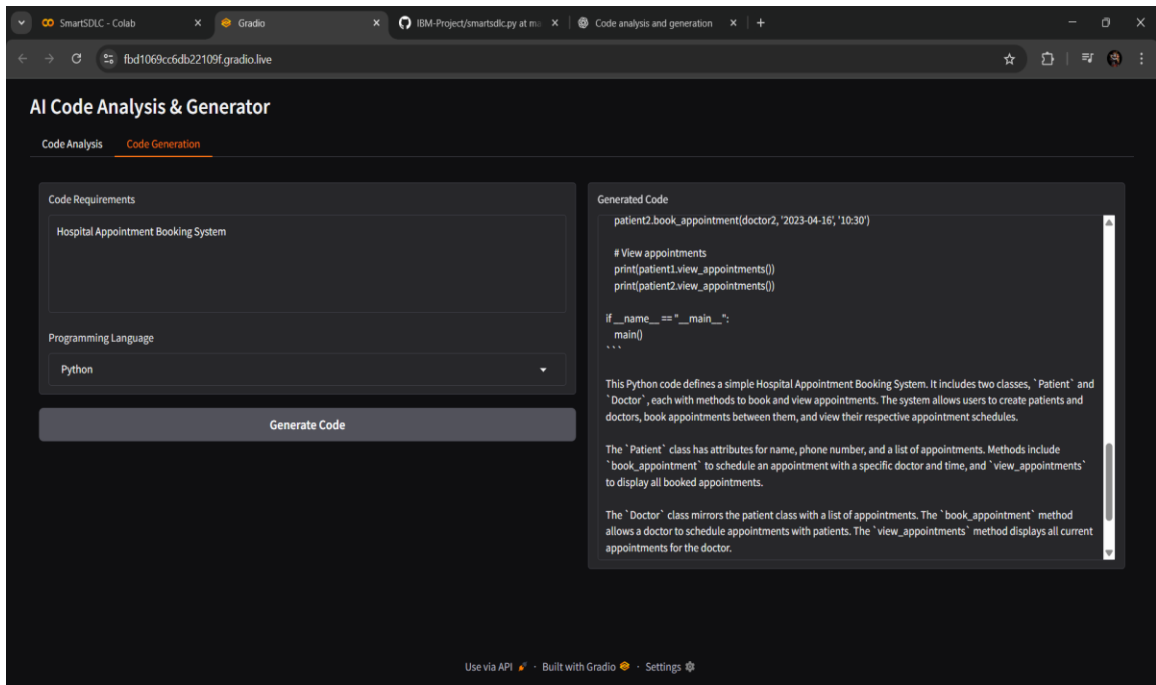
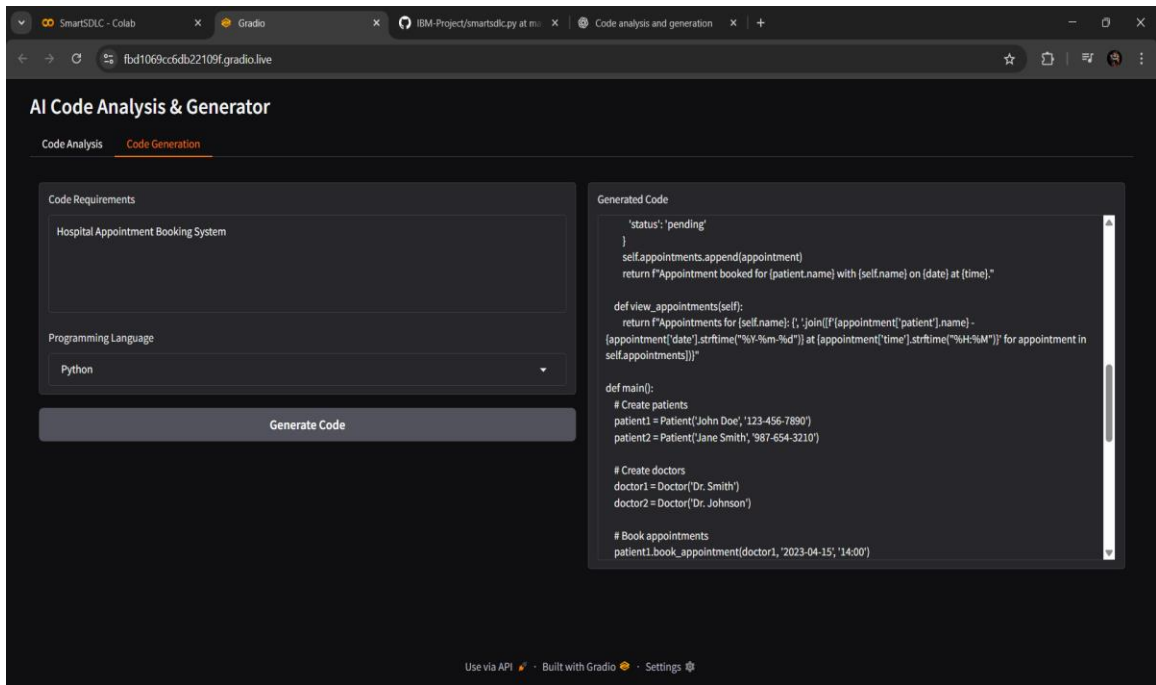












SmartSDLC - Colab

Gradio

IBM-Project/smartsdlc.py at m...

Code analysis and generation

fbf1069cc6db22109f.gradio.live

AI Code Analysis & Generator

Code Analysis

Code Generation

Code Requirements

Hospital Appointment Booking System

Programming Language

Python

Generate Code

Generated Code

```
if __name__ == '__main__':
    main()
...
```

This Python code defines a simple Hospital Appointment Booking System. It includes two classes, 'Patient' and 'Doctor', each with methods to book and view appointments. The system allows users to create patients and doctors, book appointments between them, and view their respective appointment schedules.

The 'Patient' class has attributes for name, phone number, and a list of appointments. Methods include 'book_appointment' to schedule an appointment with a specific doctor and time, and 'view_appointments' to display all booked appointments.

The 'Doctor' class mirrors the patient class with a list of appointments. The 'book_appointment' method allows a doctor to schedule appointments with patients. The 'view_appointments' method displays all current appointments for the doctor.

The 'main' function demonstrates how to use the system by creating patients, doctors, and booking appointments. It then prints the appointment schedules for both patients.

This code provides a basic structure for a hospital appointment booking system and can be extended with more features like appointment cancellation, reminders, and integration with a database or web interface.

Use via API · Built with Gradio · Settings

13. Known Issues

- PDF extraction may fail for **scanned or image-based PDFs** (OCR not yet supported).
 - Large PDFs can cause **slow processing** or **high memory usage**.
 - Generated code may need **manual debugging or validation**.
 - Responses can sometimes be **too generic or incomplete**.
 - No **authentication or user restrictions** (anyone with the link can access).
-

13. Future Enhancements

- Add **authentication and user management** (login, API keys, role-based access).
 - Integrate **OCR support** for scanned/image-based PDFs.
 - Improve **code generation** with test cases and frameworks.
 - Add **debugging and code review** features.
 - Enhance **UI/UX** with dashboards and better visualization.
 - Enable **version control** for generated code.
 - Provide **export options** (PDF, DOCX, or JSON reports).
-