SmartSDLC – AI-Enhanced Software Development

Lifecycle

# Project Documentation

# 1.Introduction

Project title :

SmartSDLC – AI-Enhanced Software Development Lifecycle

Team members:

1. Deepika V (Team leader)
2. Manjula A
3. Dharani M
4. Aswini J
5. Anupriya B

# 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](https://www.python.org/downloads/release/python-3100/)
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

1. **Install Dependencies**

* pip install torch transformers gradio PyPDF2

1. **Run the Application**

* **python app.py**

1. **In Google Colab (Recommended)**

* Upload project files to Colab.
* Install dependencies inside Colab**:**
* **!pip install torch transformers gradio PyPDF2**

1. **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).

1. **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**

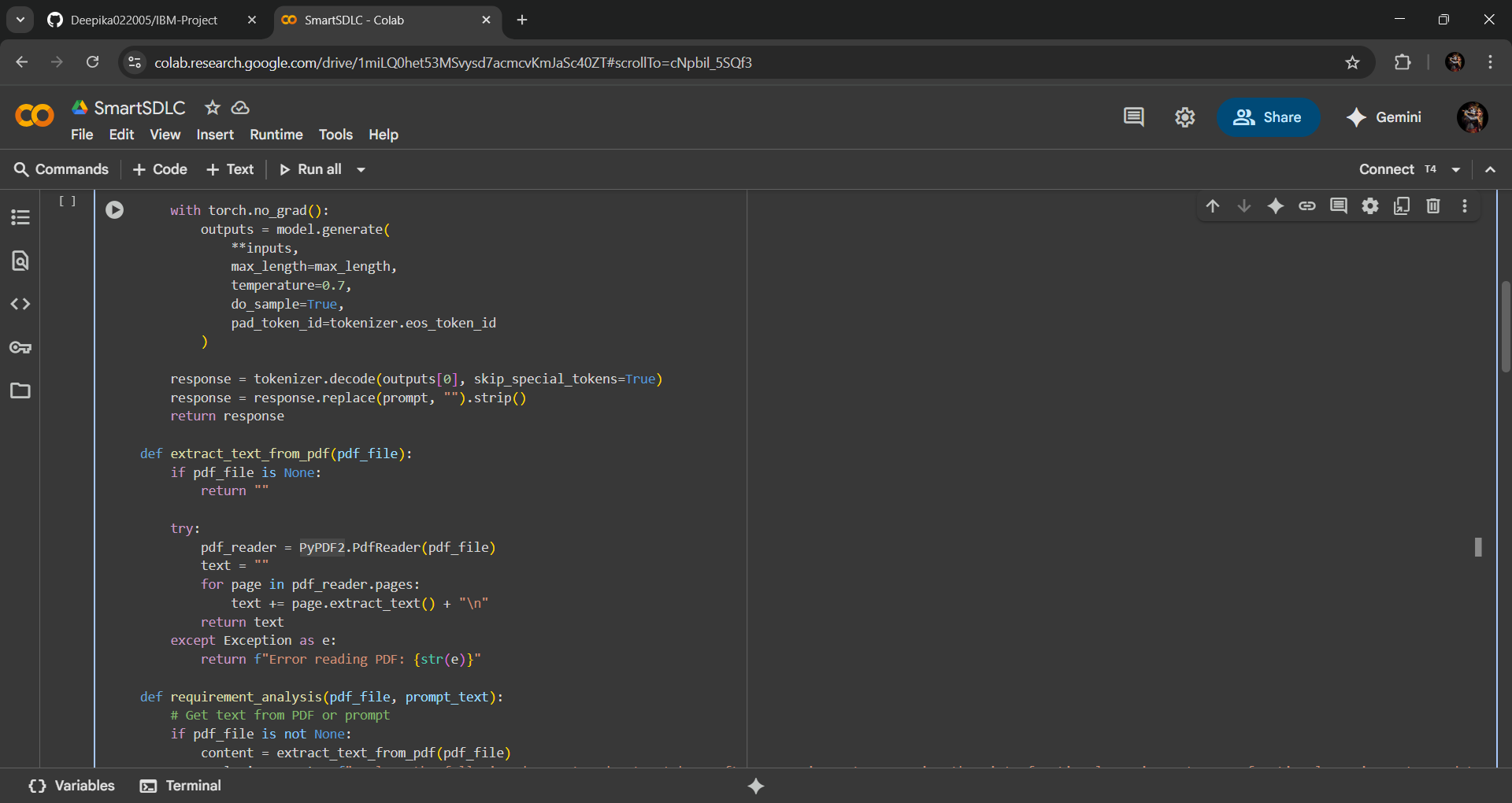
* Simple and easy-to-use layout.
* Tabs separate analysis and code generation for clarity.
* Works on desktop and mobile browsers.

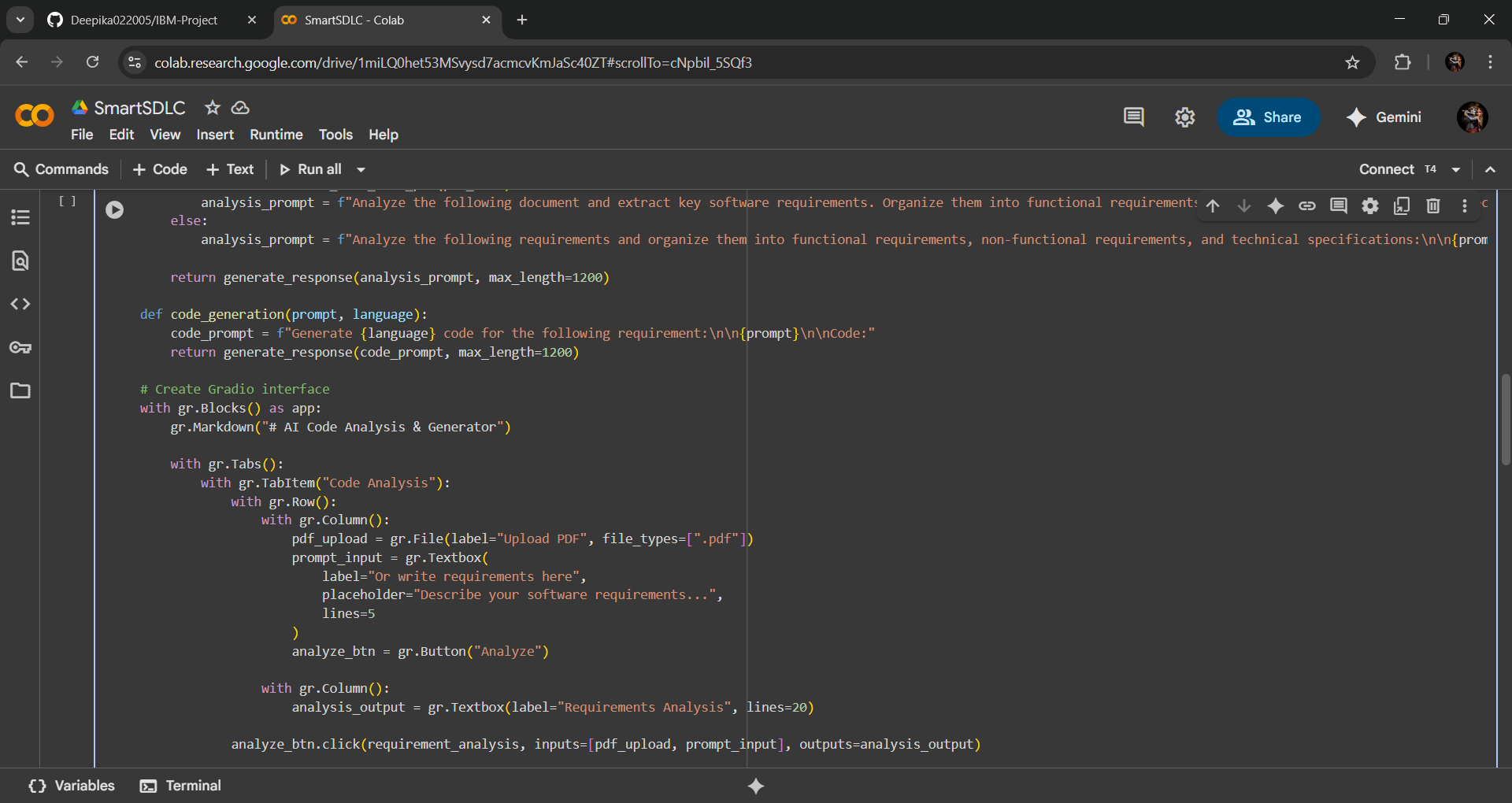
# 10. Testing

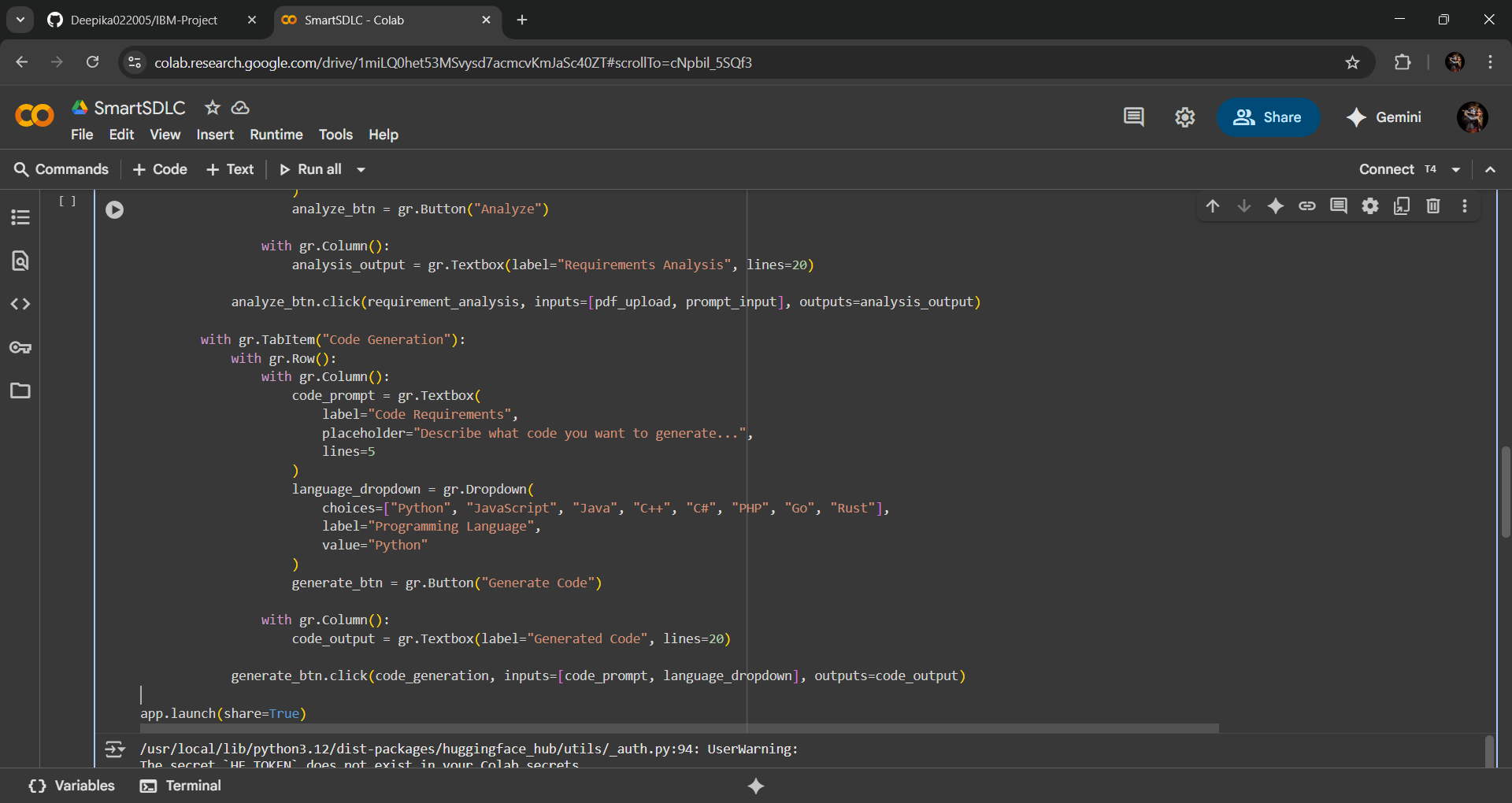
* Upload sample PDFs to test **requirement extraction**.
* Enter requirement text directly to test **text-based analysis**.
* Generate code in different programming languages to verify **code generation**.
* Check that the **Gradio interface** works correctly, including buttons, textboxes, and dropdowns.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

# 11. Source Code

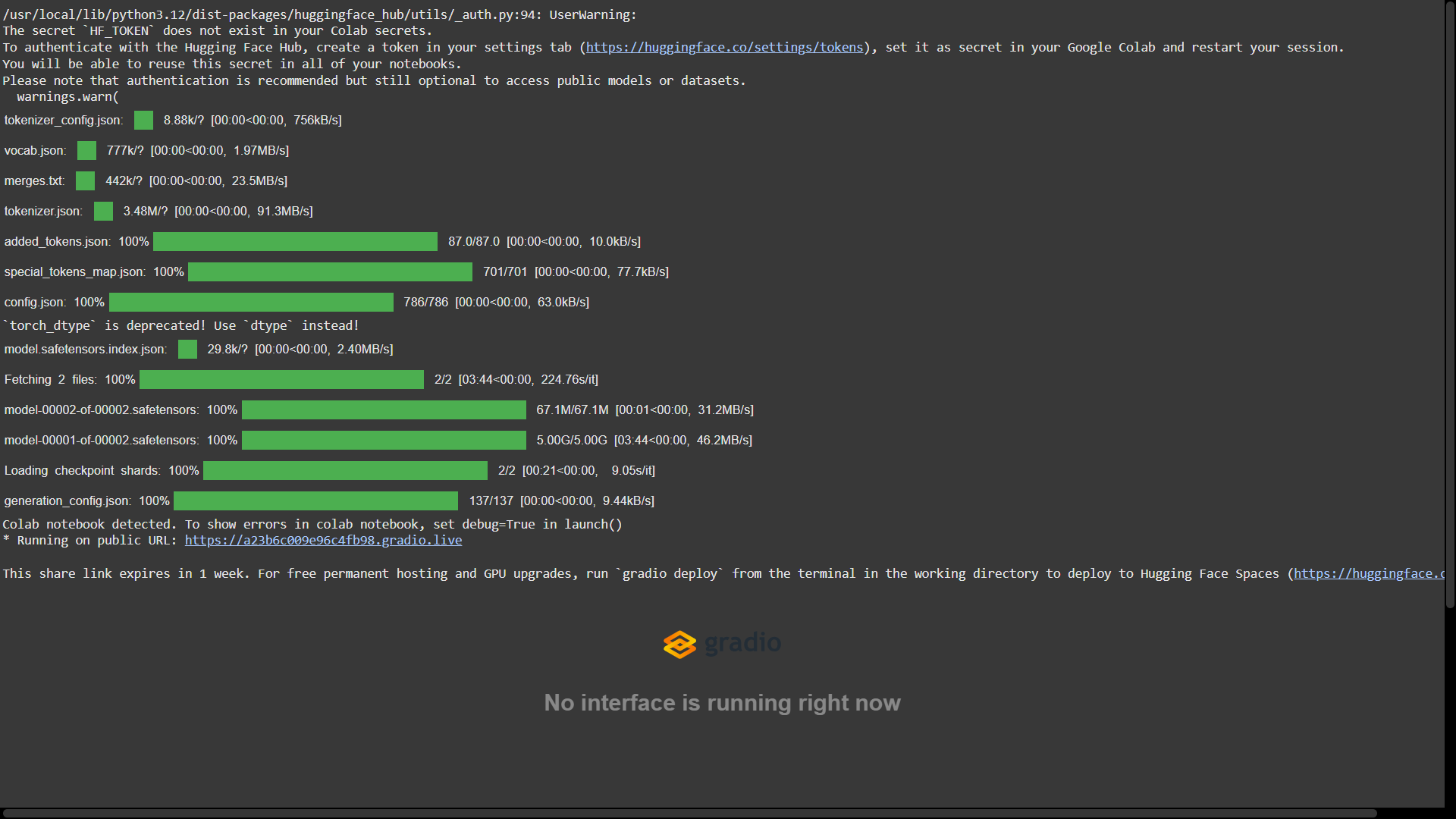
# 

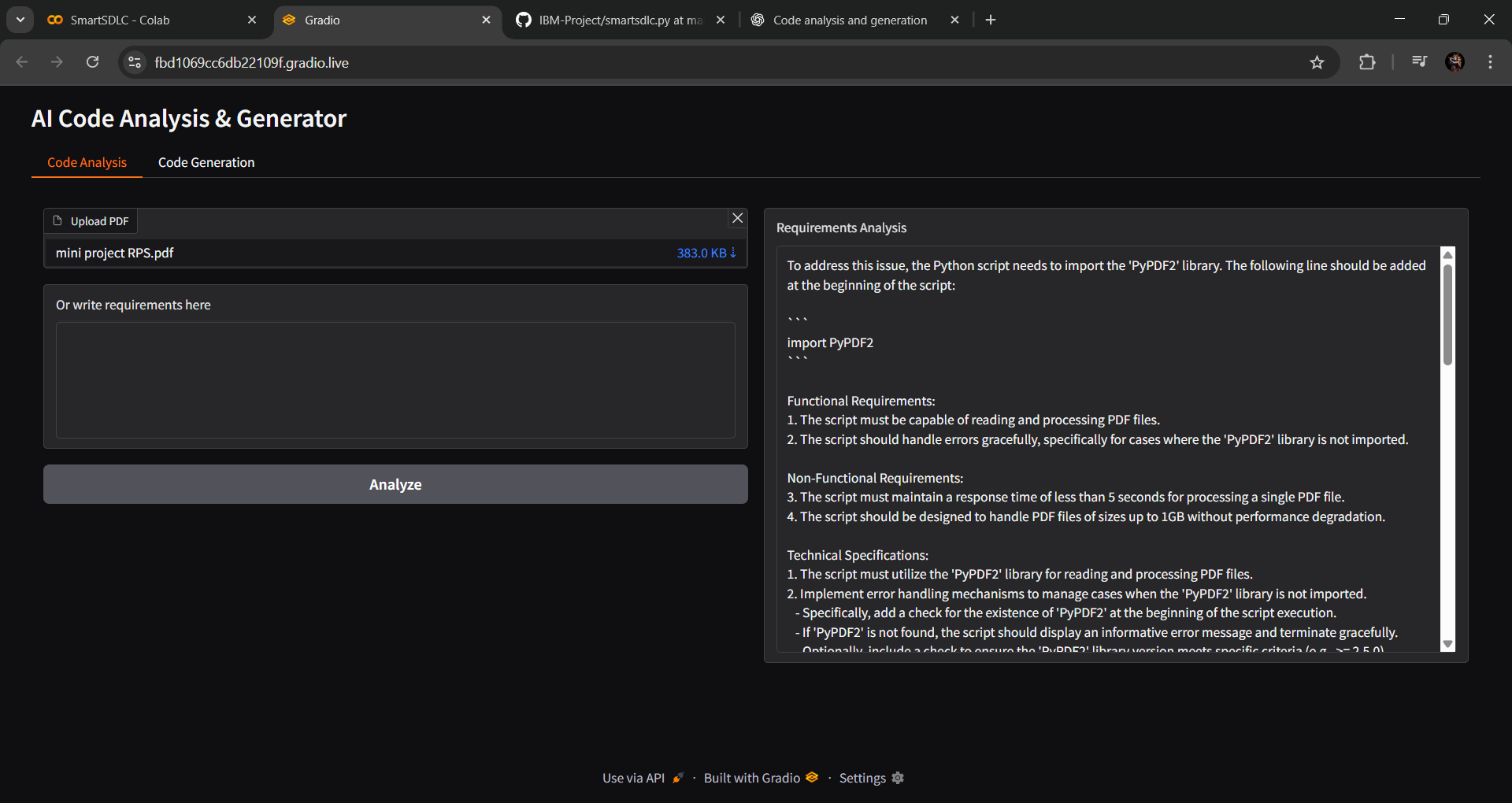
****

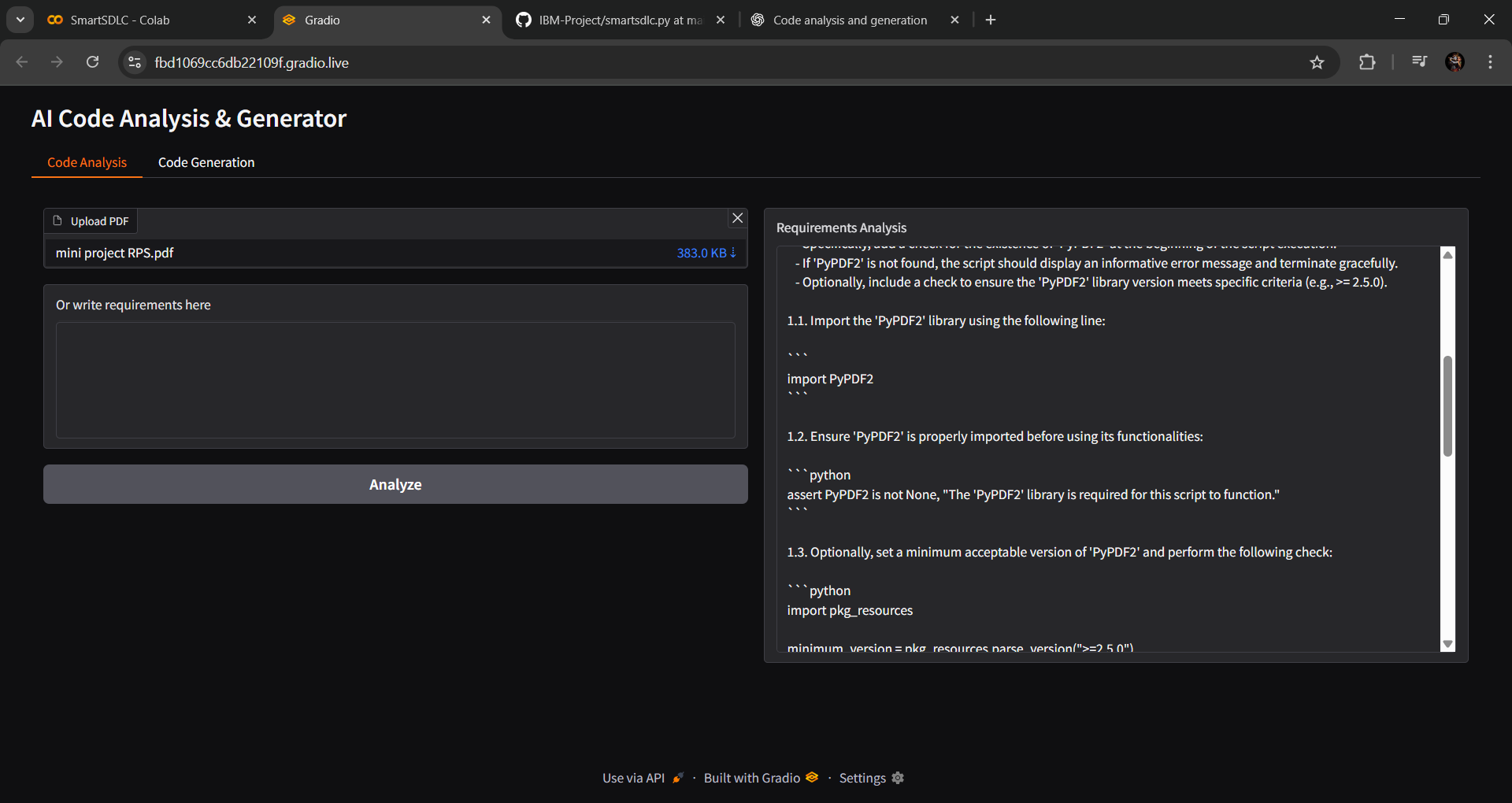
****

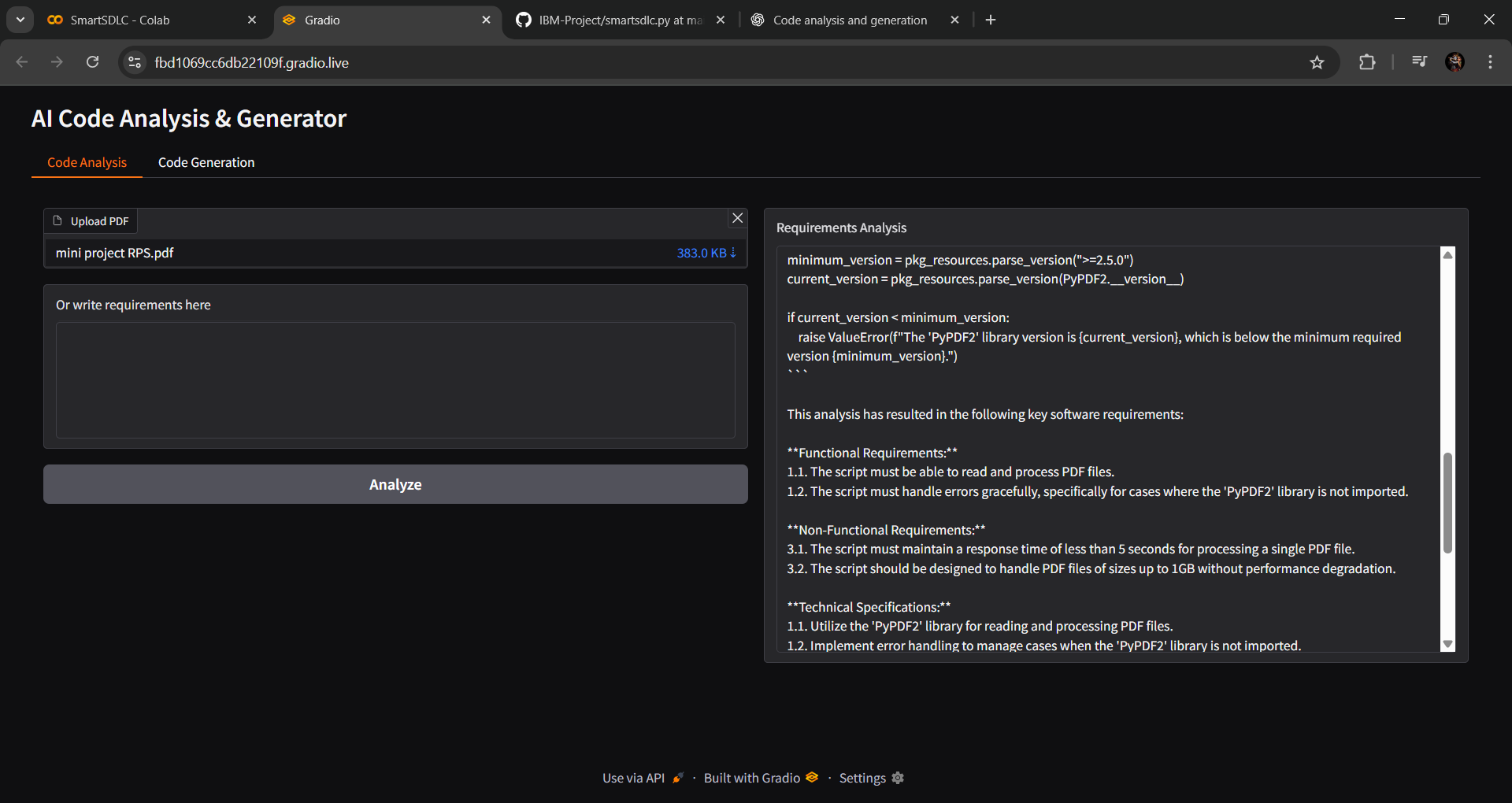
****

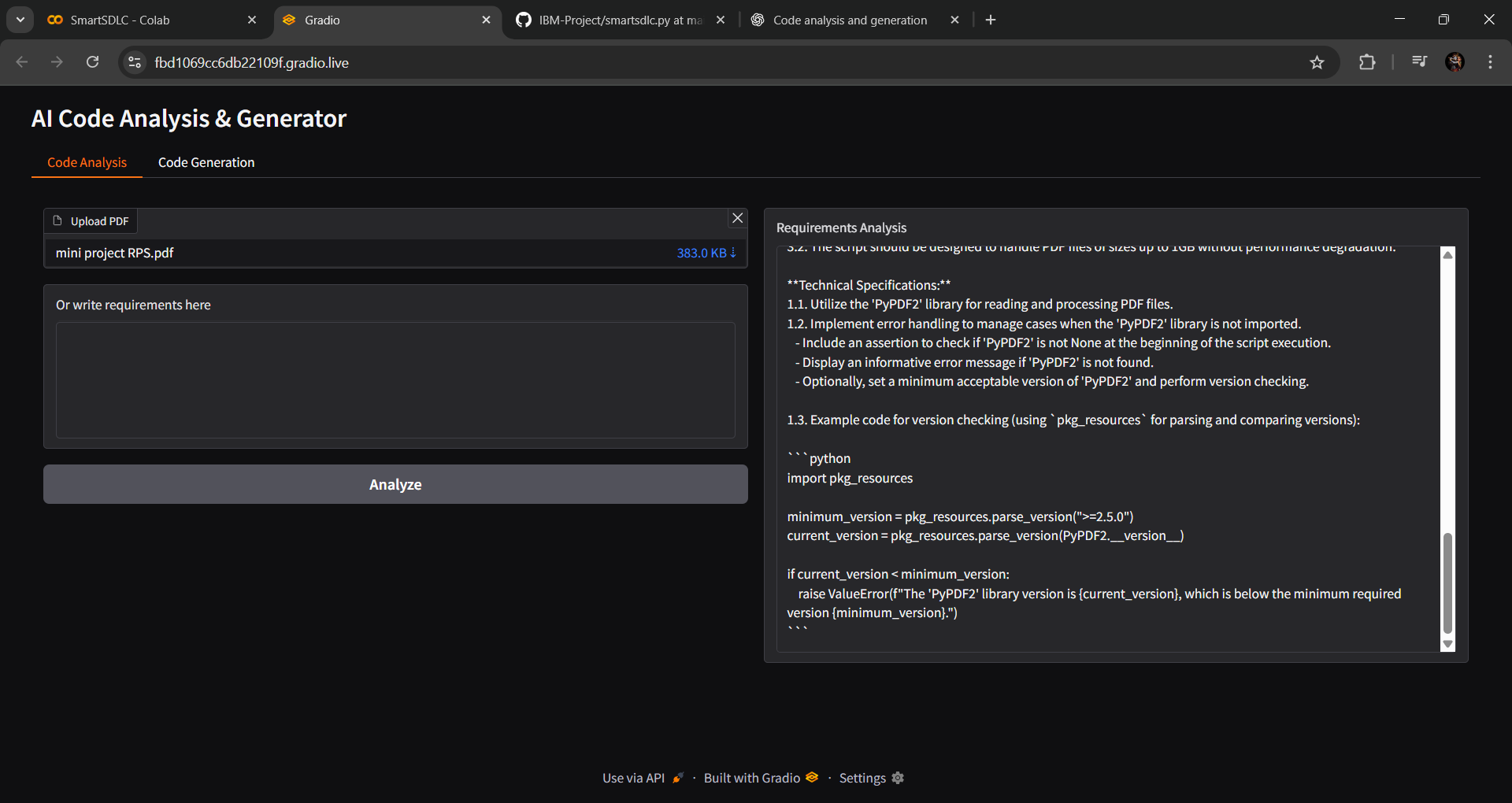
# 12. Output

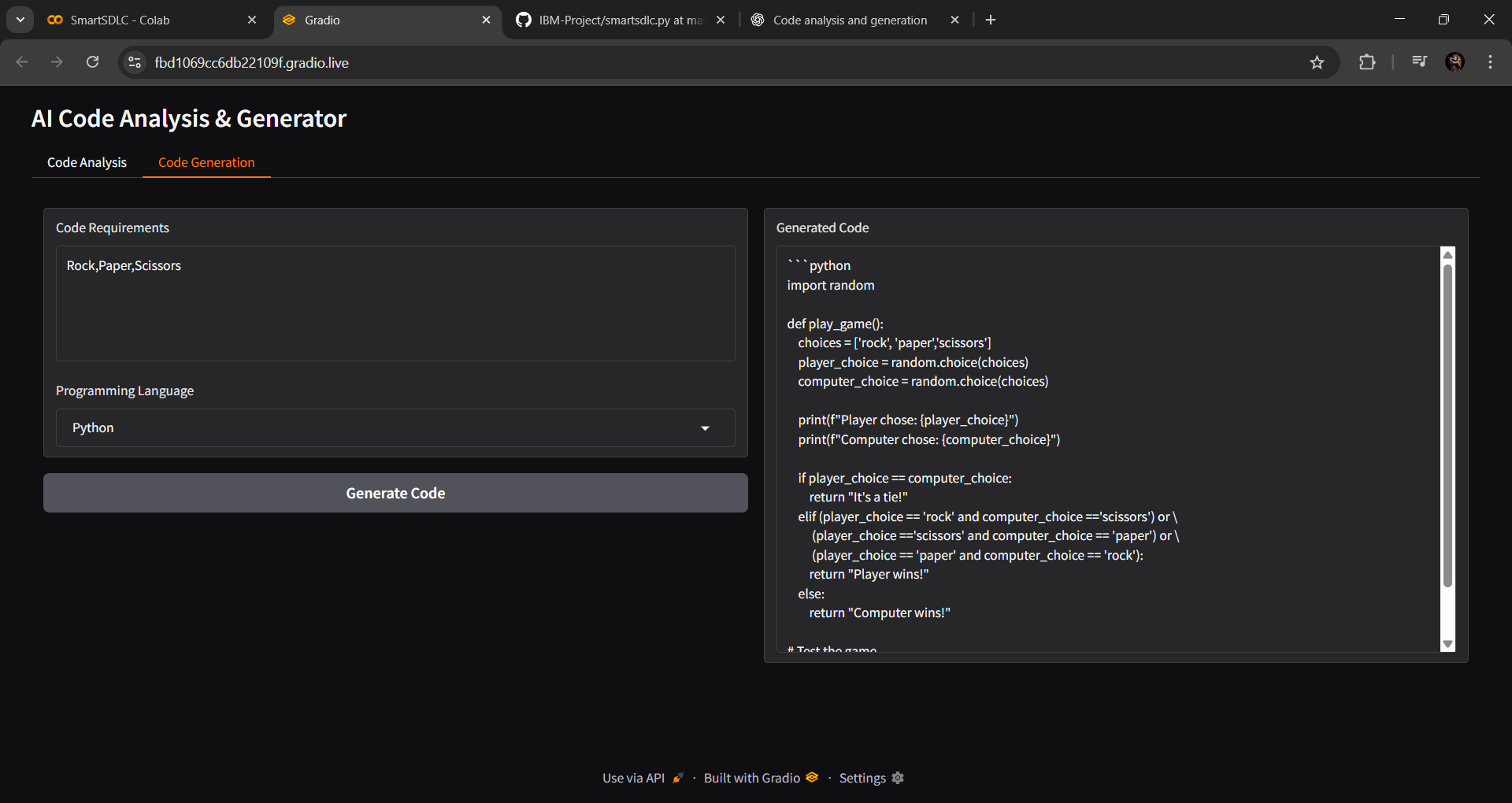
****

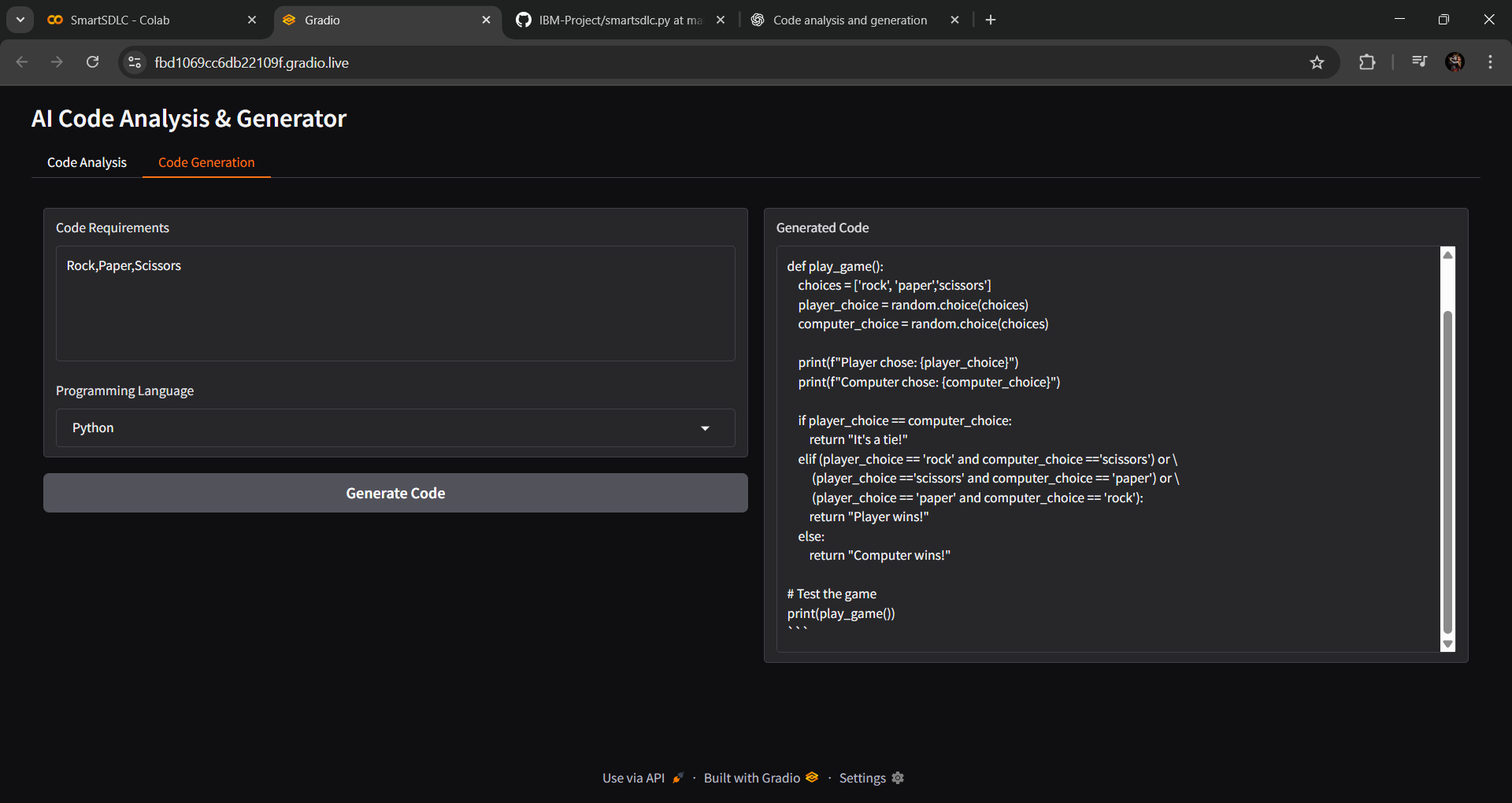
****

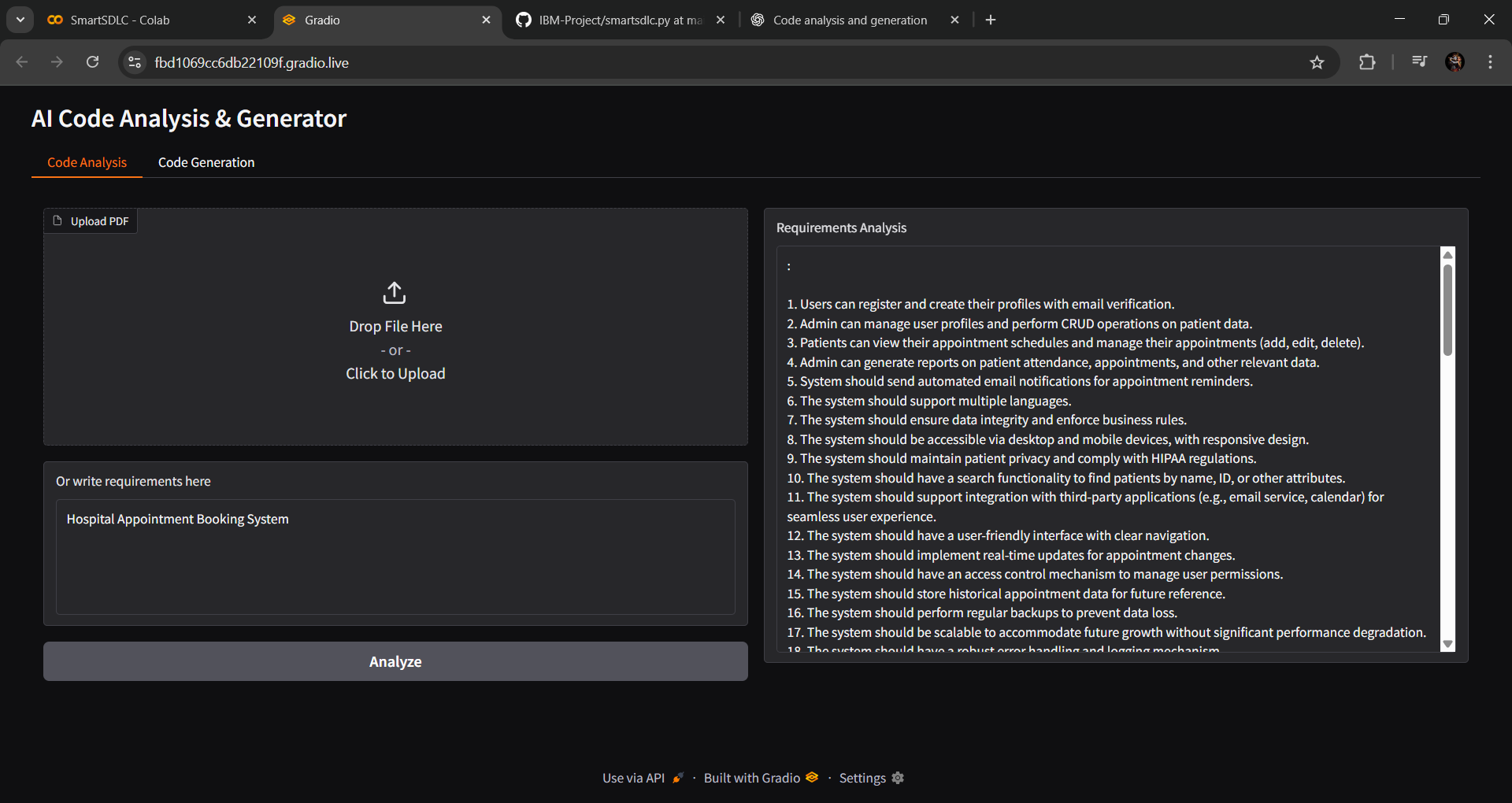
****

****

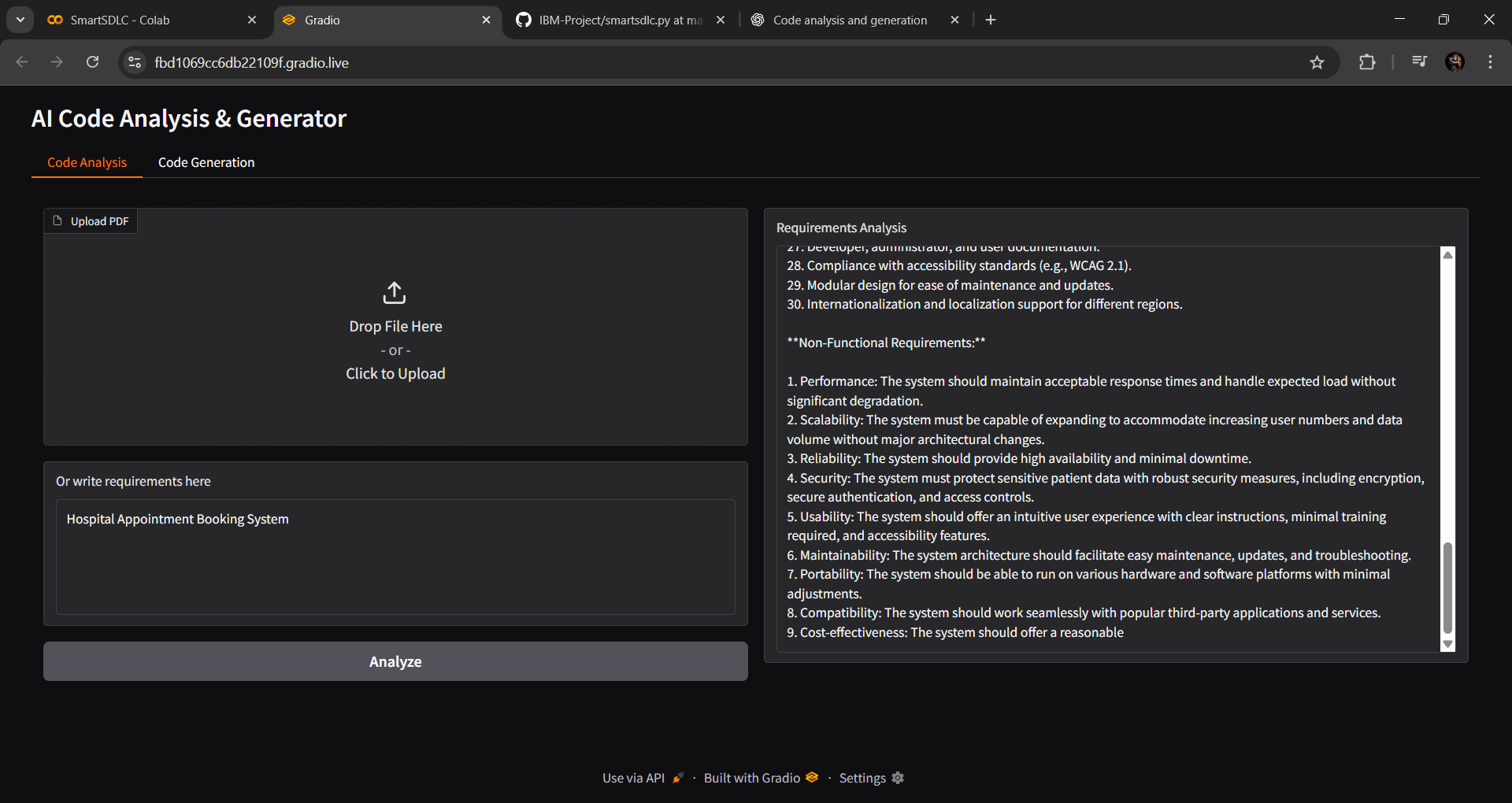
****

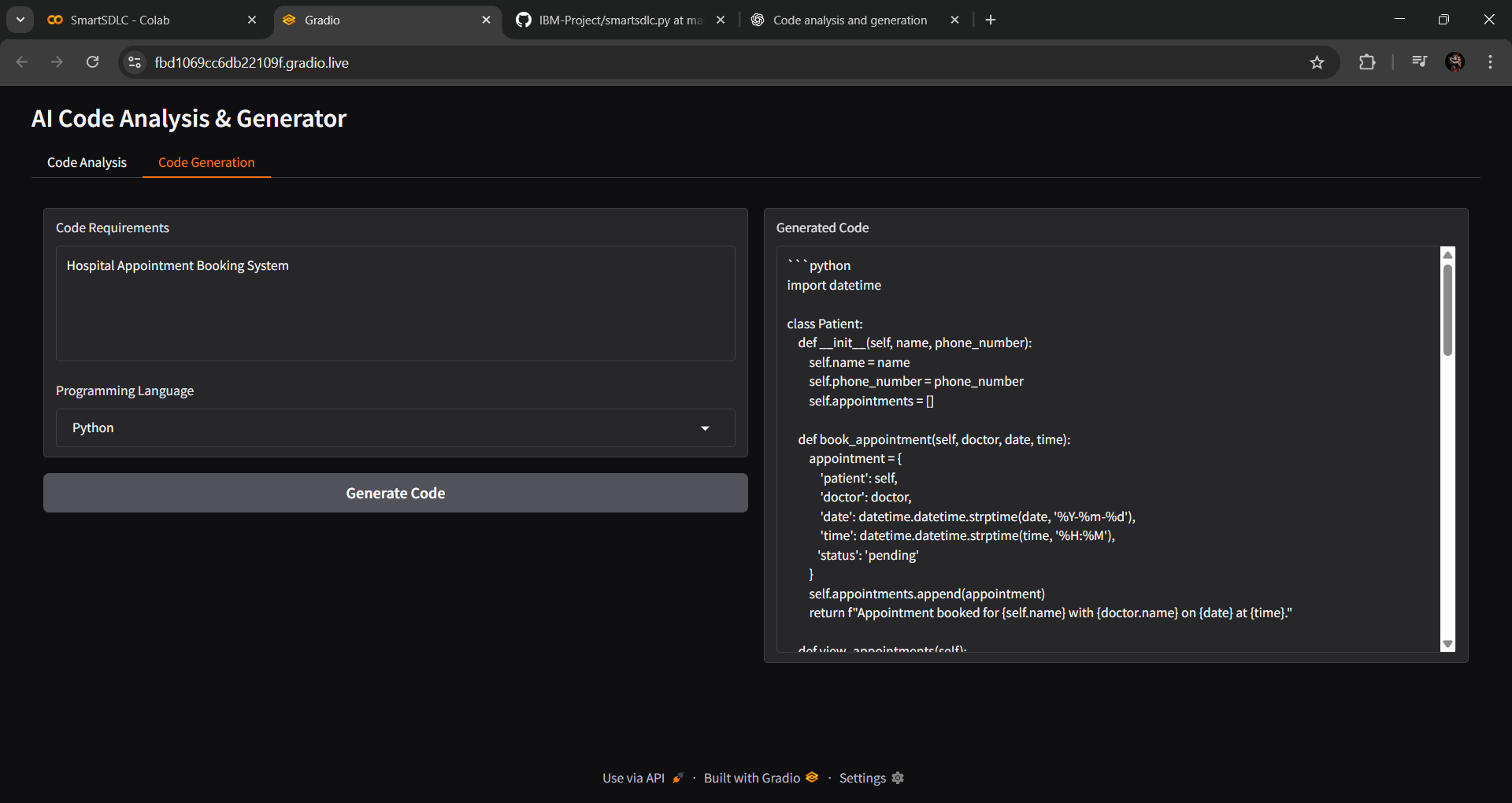
****

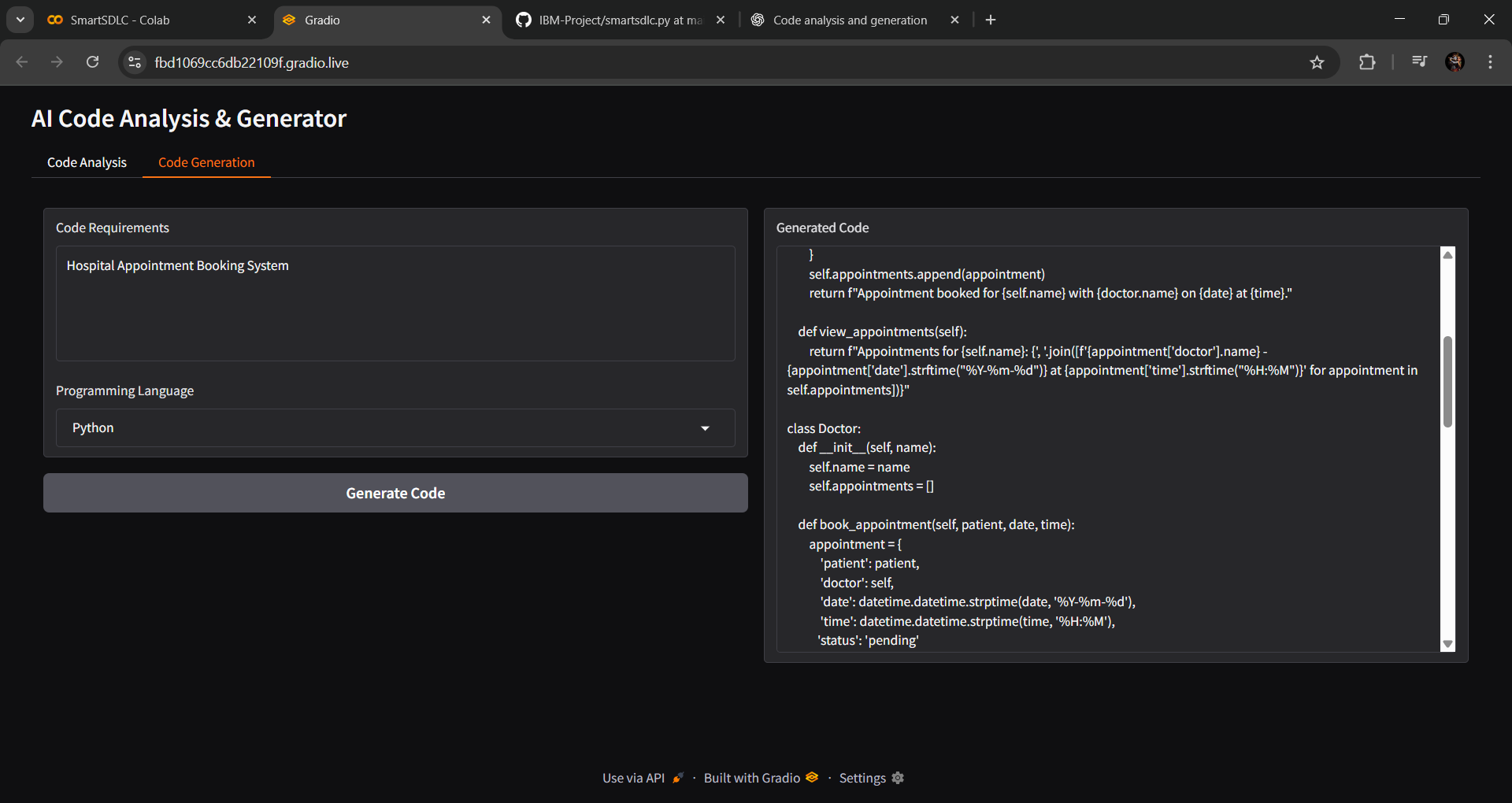
****

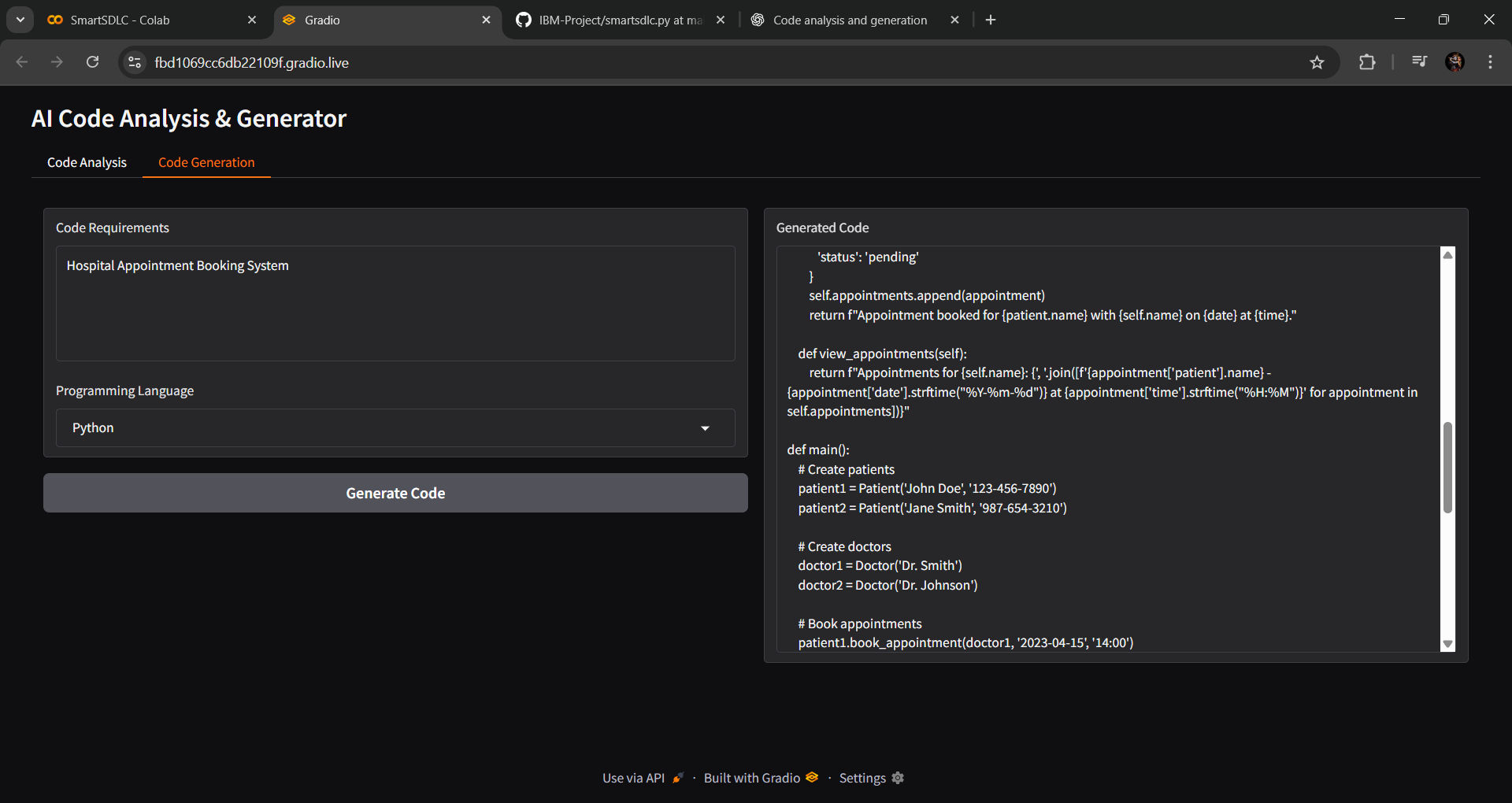
****

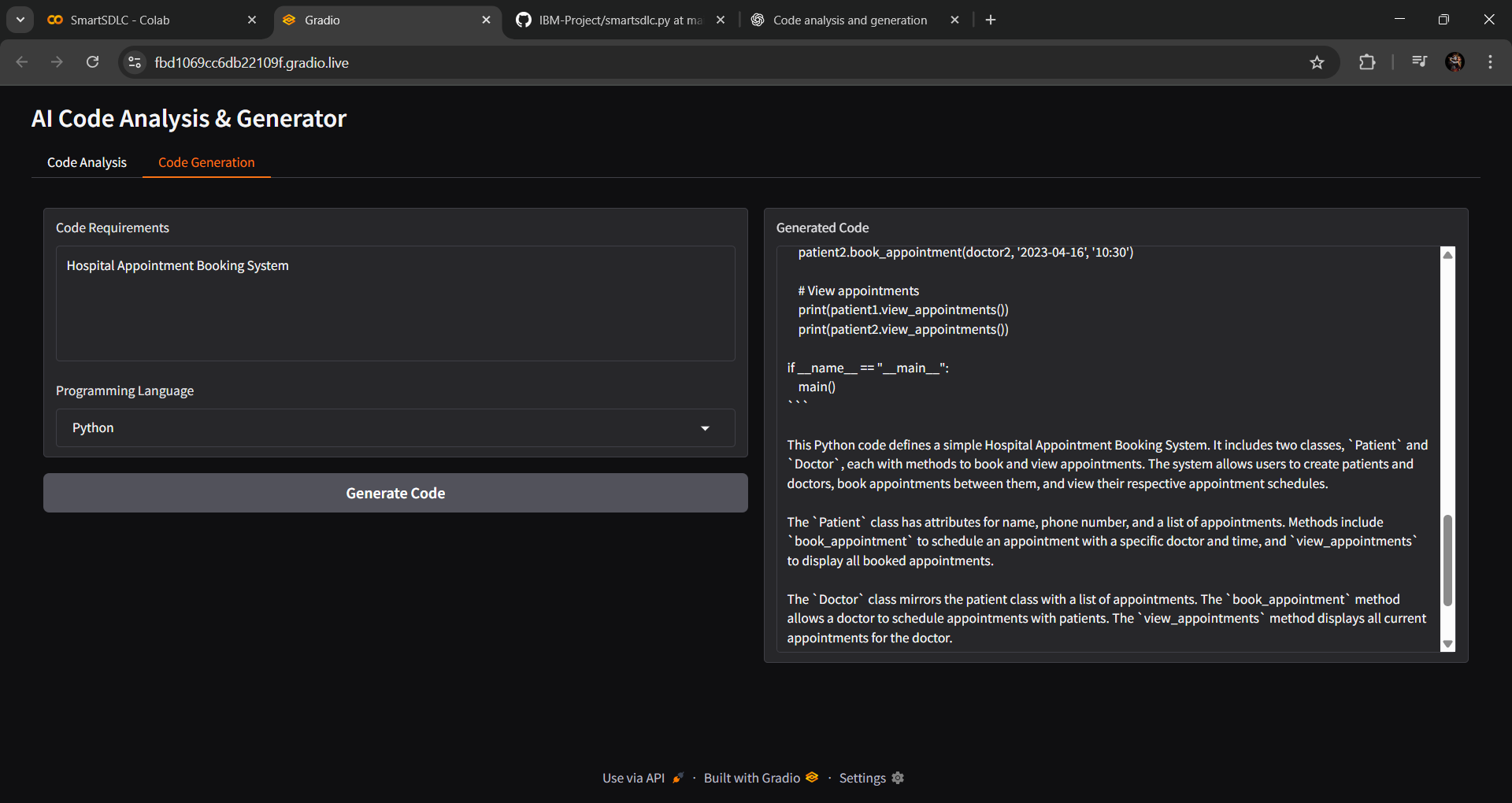
# 

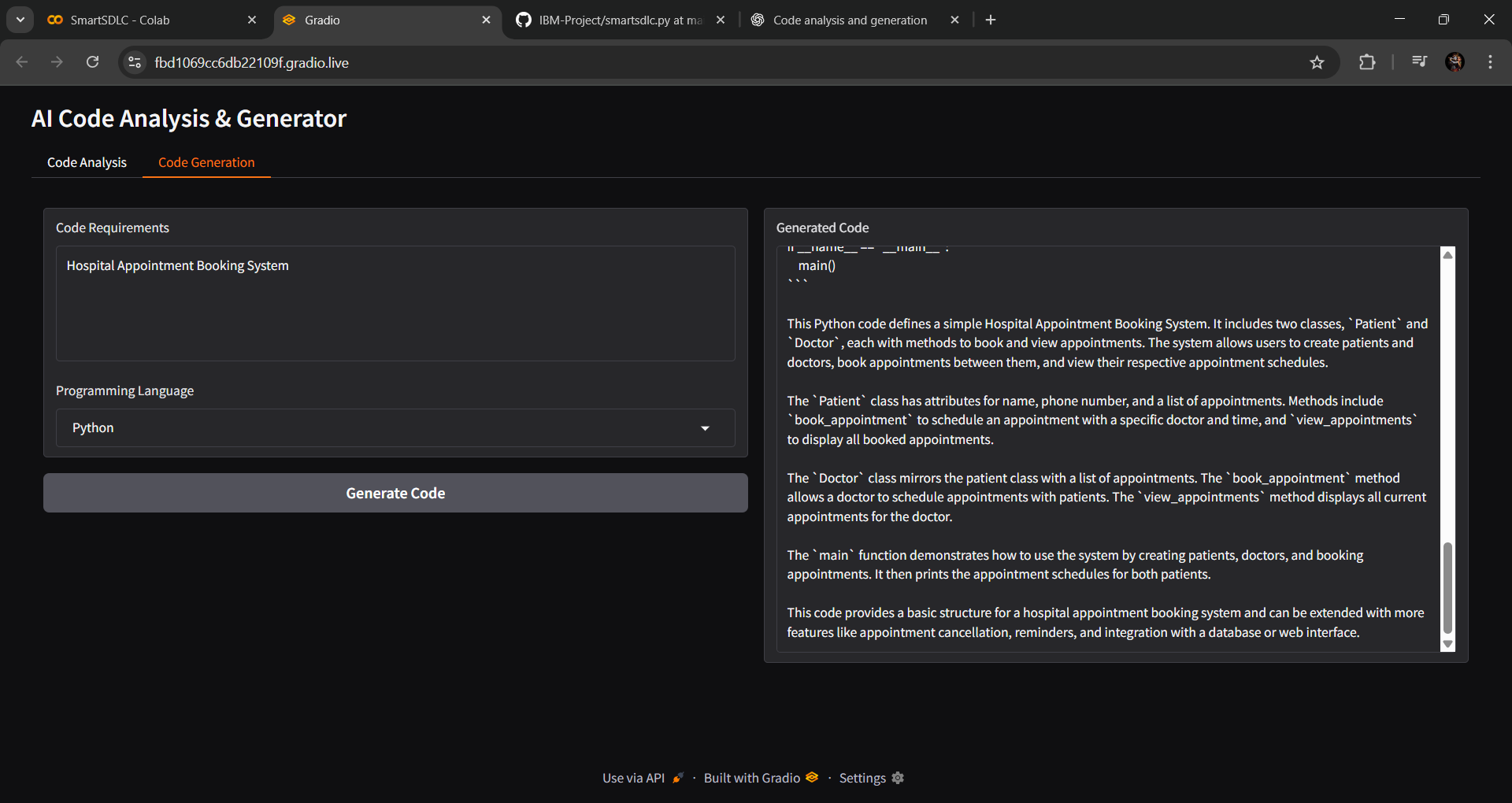
****

****

****







# 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).