SmartSDLC-AI-Enhanced Software Development Lifecycle

Project Documentation

1. Introduction

- **Project Title:** SmartSDLC – AI-Enhanced Software Development Lifecycle
- **Team Member: jasmin.R**
- **Team Member: Harshaa.G**
- **Team Member: Agalya.M**
- **Team Member: Mubina.S**

---

2. Project Overview

Purpose:
The purpose of SmartSDLC is to automate and streamline the **Software Development Lifecycle (SDLC)** by leveraging **AI, LLMs, and real-time analysis**. It enhances efficiency for developers, analysts, and project managers by providing automated requirement analysis, intelligent code generation, bug detection, and policy summarization. SmartSDLC acts as an intelligent assistant to improve productivity, reduce errors, and optimize the overall software development process.

Features:

- **Requirement Analysis**
  - *Key Point:* AI-driven requirements extraction
  - *Functionality:* Analyzes uploaded documents (PDFs, text) and organizes requirements into **functional, non-functional, and technical** categories.
- **Code Generation**
  - *Key Point:* Multi-language code synthesis
  - *Functionality:* Generates code in multiple programming languages (Python, Java, C++, etc.) based on natural language requirements.
- **Bug Fixing & Suggestions**
  - *Key Point:* Automated debugging assistance
  - *Functionality:* Identifies potential issues in given code snippets and suggests optimized fixes.
- **Policy Summarization**
  - *Key Point:* Simplified document understanding
  - *Functionality:* Converts lengthy software specifications or compliance documents into clear, actionable summaries.
- **Test Case Creation**
  - *Key Point:* Automated QA support
  - *Functionality:* Generates unit test cases from functional requirements to ensure coverage.
- **Eco Tips for Developers (Optional Extension)**

    - *Key Point:* Sustainable coding practices
    - *Functionality:* Provides tips on writing energy-efficient, optimized, and maintainable code.
- **Gradio/Streamlit UI**
    - *Key Point:* User-friendly AI interface
    - *Functionality:* Provides an intuitive dashboard with **code analysis, requirement upload, and generation features**.

---

3. Architecture

- **Frontend (Gradio / Streamlit):**
  Provides an interactive web interface with tabs for **requirement analysis, code generation, and bug fixing**. Users can upload PDFs, type requirements, and view AI outputs in real-time.
- **Backend (FastAPI):**
  Serves as the REST framework that handles **API requests for analysis, generation, summarization, and testing**.
- **LLM Integration (IBM Watsonx Granite):**
  Uses **Granite LLM models** for natural language understanding, code synthesis, and summarization.
- **Document Processing (PyPDF2):**
  Extracts text from uploaded requirement PDFs for analysis.
- **ML Modules (Bug Detection & Optimization):**
  Machine learning models analyze and detect

common errors, inefficiencies, and suggest optimizations.

---

4. Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip & virtual environment
- IBM Watsonx API key
- GPU (optional for faster inference)

Installation:

1. Clone repository
2. Install dependencies:
3. pip install -r requirements.txt
4. Configure .env file with API credentials
5. Run backend with FastAPI
6. Launch frontend using Gradio/Streamlit

---

5. Folder Structure

app/ – FastAPI backend (routes, models, utils)

app/api/ – Modular APIs (chat, code, requirements, feedback)

ui/ – Frontend pages (Gradio/Streamlit components)

smart_sdlc.py – Entry script for launching dashboard granite_llm.py – Handles IBM Watsonx Granite LLM interactions

pdf_processor.py – Extracts and preprocesses document text

requirement_analyzer.py – Organizes requirements (FR/NFR/Tech)

code_generator.py – Generates code in multiple languages

test_case_generator.py – Creates automated test cases

bug_fixer.py – Suggests fixes for code issues

---

6. Running the Application

1. Start FastAPI backend
2. Run Gradio/Streamlit dashboard
3. Navigate through UI tabs:
    - Upload requirements (PDF/text)
    - Generate code in selected language
    - Analyze and fix code bugs
    - View summarized requirements & reports

---

7. API Documentation

- **POST /analyze-requirements** – Extracts functional/non-functional/technical requirements

- **POST /generate-code** – Generates code in chosen language
- **POST /bug-fix** – Suggests bug fixes and optimizations
- **POST /test-cases** – Produces test cases from requirements
- **POST /summarize-policy** – Summarizes large documents

---

8. Authentication

For demonstration, runs open. Secure deployment can use:

- API Keys (IBM Watsonx)
- JWT Token-based Authentication
- Role-based Access (Admin/Developer/Tester)

---

9. User Interface

- Minimalist Gradio/Streamlit dashboard
- Tabs for **Requirement Analysis, Code Generation, Bug Fixing**
- File upload support for PDFs
- Code editor-like output box for generated code
- Downloadable reports

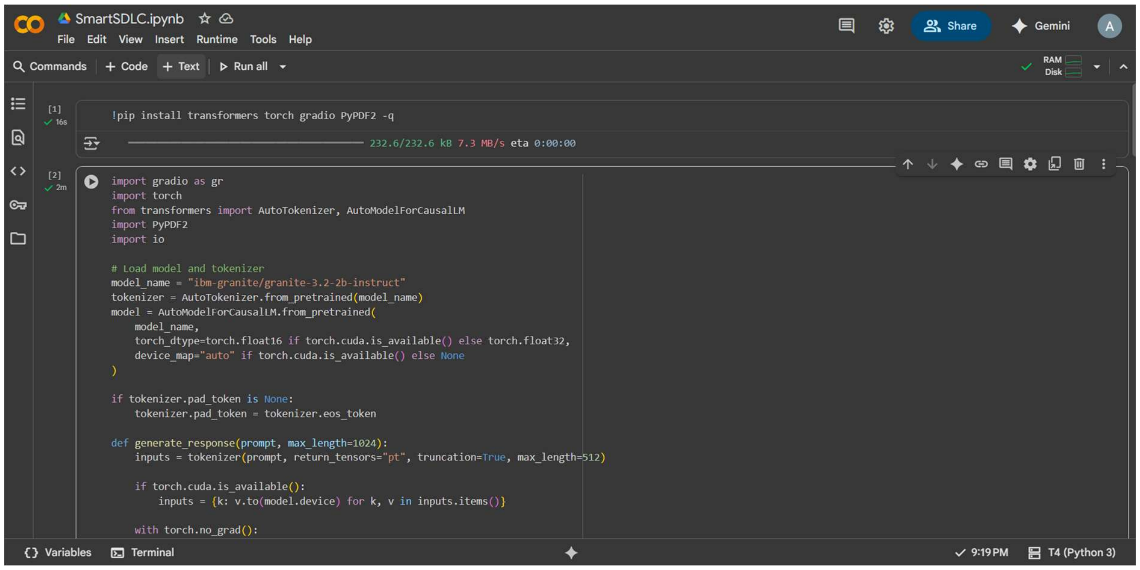
---

10. Testing

- **Unit Testing:** Prompt functions and utilities
- **API Testing:** Swagger, Postman
- **Manual Testing:** Requirement uploads, code outputs
- **Edge Cases:** Empty files, large PDFs, invalid inputs

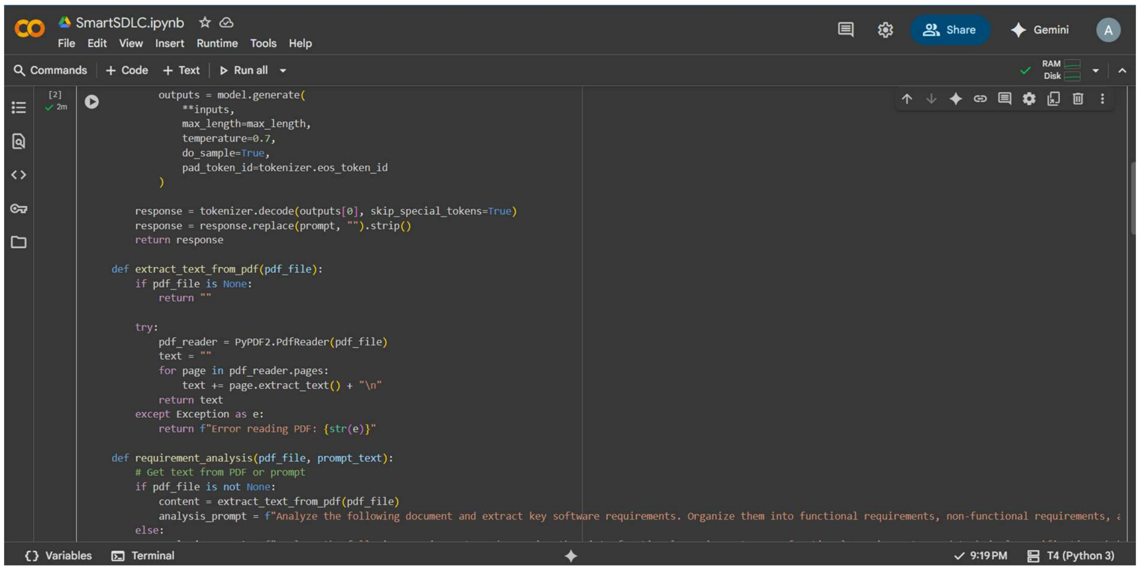
---

11. Screenshots

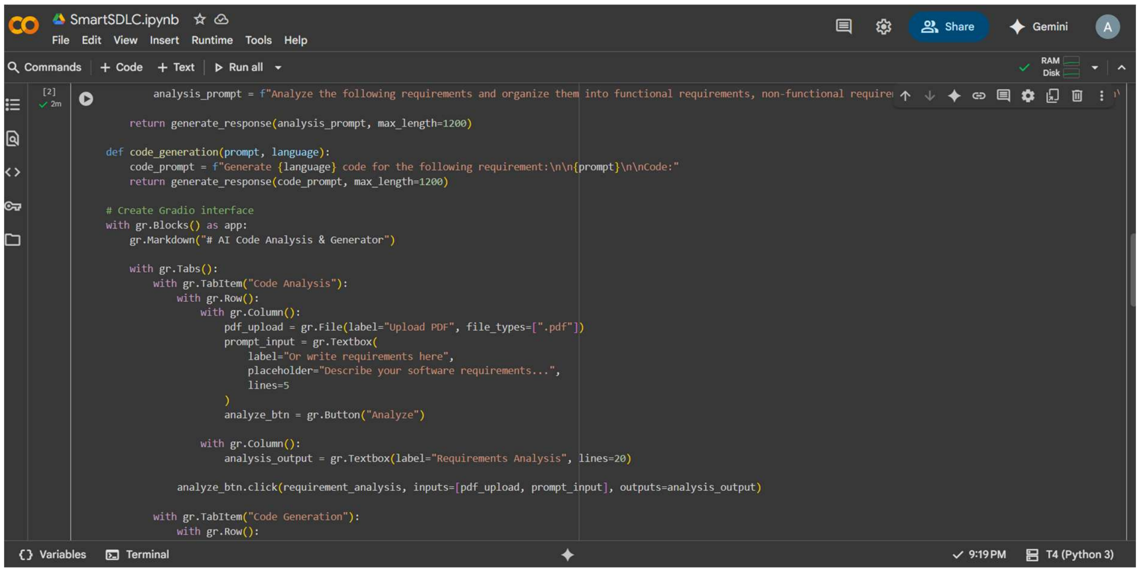


```
!pip install transformers torch gradio PyPDF2 -q
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 7.3 MB/s eta 0:00:00
```

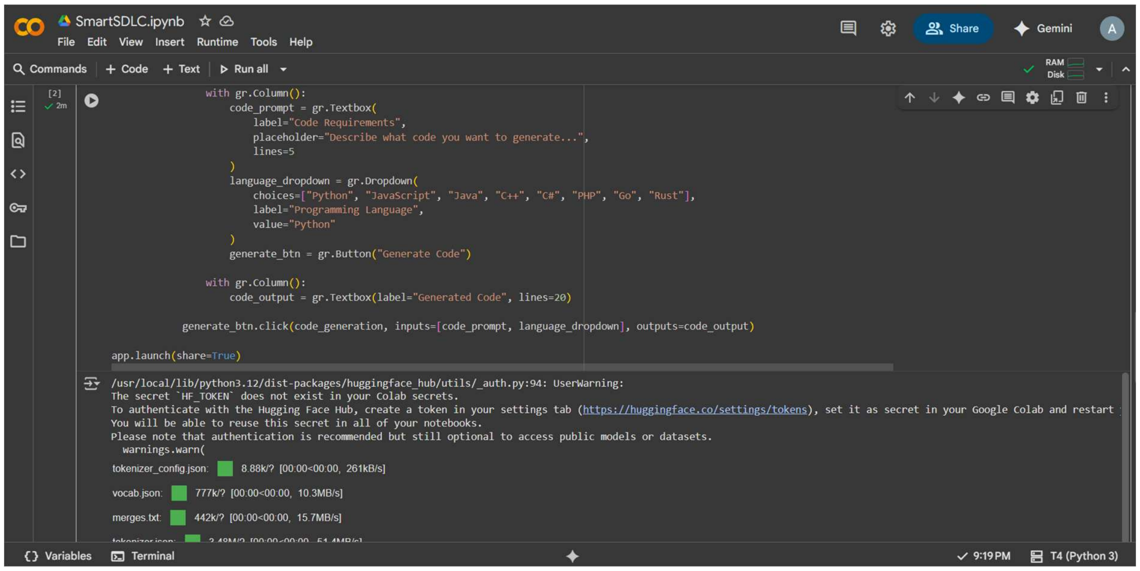
```python
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io

# 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():
```



```python
        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)
        analysis_prompt = f"Analyze the following document and extract key software requirements. Organize them into functional requirements, non-functional requirements, a
    else:
```

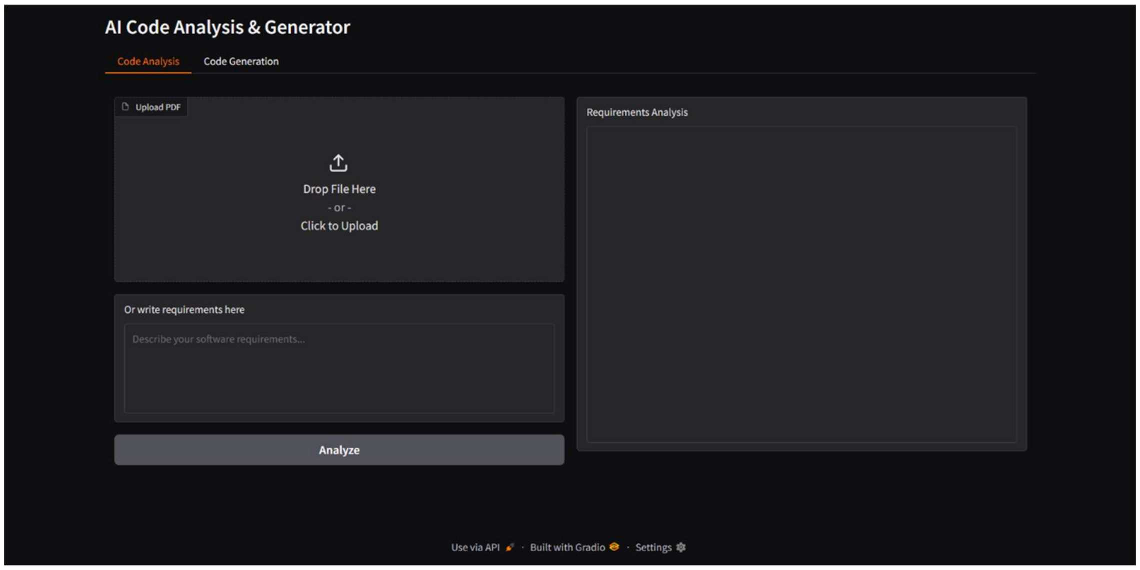

SmartSDLC.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
Variables Terminal 9:19 PM T4 (Python 3)

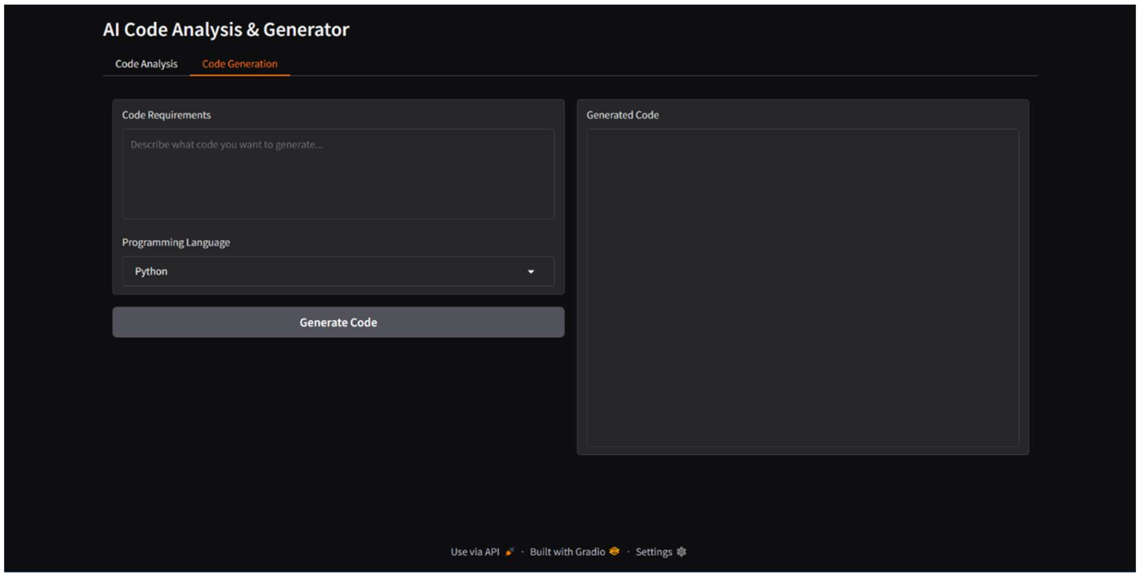
```
        analysis_prompt = f"Analyze the following requirements and organize them into functional requirements, non-functional require

    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)

        with gr.TabItem("Code Generation"):
            with gr.Row():
```


SmartSDLC.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
Variables Terminal 9:19 PM T4 (Python 3)

```
                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.
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
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, 261kB/s]
vocab.json: 🟩 777k/? [00:00<00:00, 10.3MB/s]
merges.txt: 🟩 442k/? [00:00<00:00, 15.7MB/s]
```


AI Code Analysis & Generator
Code Analysis
Code Generation
Upload PDF
Drop File Here
- or -
Click to Upload
Or write requirements here
Describe your software requirements...
Analyze
Requirements Analysis
Use via API · Built with Gradio · Settings


AI Code Analysis & Generator
Code Analysis
Code Generation
Code Requirements
Describe what code you want to generate...
Generated Code
Generate Code
Use via API · Built with Gradio · Settings

Programming Language
Python

12. Known Issues

- Large PDF parsing may take longer
- Generated code may require manual optimization
- Limited offline functionality (requires API access)

13. Future Enhancements

- Integration with **GitHub/GitLab** for direct code commits
- **CI/CD pipeline support**
- Advanced debugging with runtime execution feedback
- Expanded test case coverage with automated test runners
- Team collaboration dashboard with project tracking