FINAL REPORT:

PROJECT TITLE:

SMART SDLC - AI - Enhanced Software Development LifeCycle

- **Problem statement:**

  Traditional software development processes within the Software Development Lifecycle (SDLC) involve significant manual effort, time consumption, and human error at various stages such as requirements analysis, code generation, testing, and documentation. Developers often struggle with interpreting unstructured requirements, creating accurate code and test cases, and maintaining consistent documentation — all of which can lead to delays, increased costs, and reduced software quality.

- **Project Goal:**

  SmartSDLC aims to transform the traditional Software Development Lifecycle (SDLC) by integrating AI technologies — specifically IBM Granite models — to automate, guide, and enhance various phases of software development. The project reduces manual effort, accelerates delivery, and improves quality through intelligent automation.

- **Project Architecture:**

  The Smart SDLC architecture provides a seamless integration between a user-facing interface, AIpowered backend processing, and deployment mechanisms. It focuses on improving efficiency across the entire software development lifecycle using IBM's generative AI capabilities.

- **Pre-requisites:**

  Python – Core language used for scripting and application logic.

  FastAPI – Backend framework used for building efficient APIs.

  Streamlit / Gradio – Lightweight frontend UI libraries for interaction.

  IBM Granite Models – Core generative model used for intelligent SDLC assistance.

- **Model Selection and Architecture:**

  Smart SDLC is a Generative AI application powered by the IBM Granite-3.3-2b-instruct model, developed to optimize the entire software development lifecycle (SDLC). The model is accessed through Hugging Face's Transformers library and utilized with PyTorch, allowing for highperformance text generation. Depending on system capabilities, it dynamically supports both CPU and GPU (with FP16 precision for GPUs).Research and select - the appropriate generative AI model suitable for SDLC use cases.Define the architecture - of the application including frontend, backend, and model integration layers.Set up the development environment - with all necessary dependencies, GPU support, and libraries (e.g., Hugging Face Transformers, Torch, Pillow).

- **Core Functionalities Development:**
  Develop the core functionalities -
  - That allow the user to interact by selecting SDLC phases
    and entering task prompts.
  - Pdf/prompt based requirement analysis.
  - Ai code generation: generate code for frontend and backend based on user prompt.

  Implement the FastAPI backend - to manage routing and user input processing, ensuring smooth API interaction and prompt handling for the Granite model

- **Main.py Development :**
  The main.py file integrates all backend logic and AI model interaction.
  Write the main application logic - including model loading, prompt formatting, AI inference, image generation, and Gradio UI logic.
  This script includes all imports, interface setup, and launching logic for local testing or public sharing and the code is :

App.py

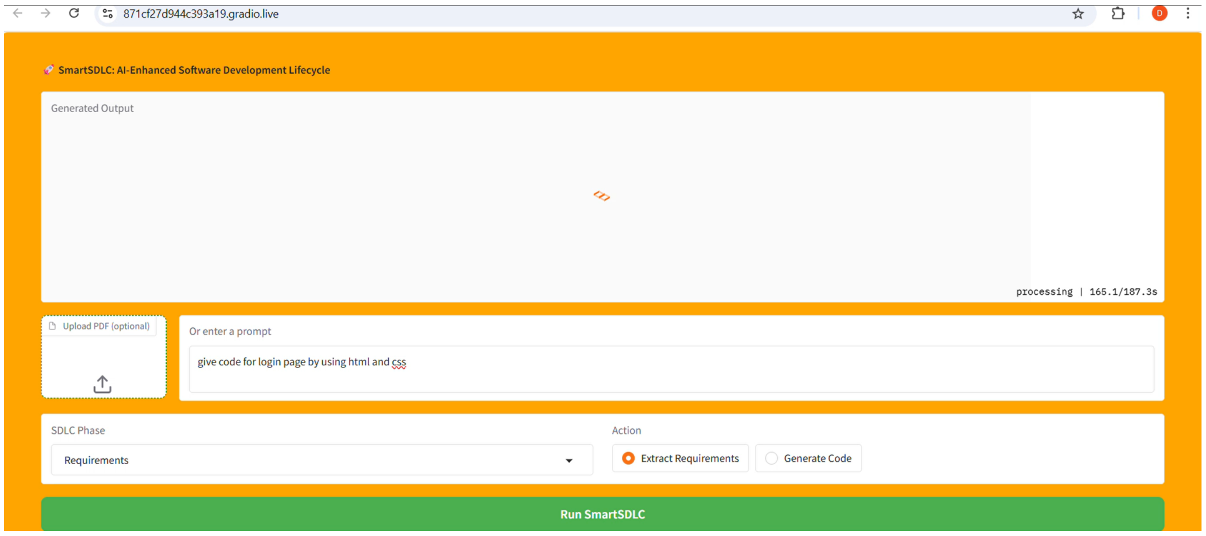
```
# -*- coding: utf-8 -*-
"""Untitled0.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1Xx293rA2OXHF49fqybjAdZY5Mu7MQ6kl
"""
pip install pymupdf

import gradio as gr
import fitz  # PyMuPDF for PDF extraction
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load IBM Granite model (replace with actual IBM Granite model name if different)
```

```
tokenizer = AutoTokenizer.from_pretrained("ibm-granite/granite-3.3-2b-instruct")
model = AutoModelForCausalLM.from_pretrained("ibm-granite/granite-3.3-2b-instruct")

def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
    for page in doc:
        text += page.get_text()
    return text

def smart_sdlc_run(pdf_file, prompt, sdlc_phase, language, framework, action):
    input_text = ""
    if pdf_file:
        input_text = extract_text_from_pdf(pdf_file.name)
    elif prompt:
        input_text = prompt
    else:
        return "Please upload a PDF or enter a prompt."

    full_prompt = f"{action} for {sdlc_phase} using {language} and {framework}: {input_text}"

    inputs = tokenizer(full_prompt, return_tensors="pt")
    outputs = model.generate(inputs["input_ids"], max_length=200, num_return_sequences=1)
    result = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return result

with gr.Blocks(css="""
.gradio-container {
  background-color: orange;
  padding: 20px;
  border-radius: 10px;
}
.textbox-bordered {
  border: 4px solid green;  /* Dark blue thick border */
  border-radius: 6px;
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.2);
}
.custom-run-button {
  background-color: #4CAF50 !important;  /* Green button */
  color: white !important;
  font-weight: bold;
  border-radius: 8px;
  padding: 10px 20px;
}
.small-file-btn {
  max-width: 150px;
  max-height: 105px;
  border-radius: 8px;
  border: 150px solid green;
}
""") as demo:
    gr.Markdown("🚀 **SmartSDLC: AI-Enhanced Software Development Lifecycle**")

    with gr.Row():
        output_text = gr.Textbox(label="Generated Output", lines=10, elem_classes=["textbox-
bordered"])

    with gr.Row():
        file_input = gr.File(label="Upload PDF (optional)", file_types=['.pdf'], type="filepath",
elem_classes=["small-file-btn"])
        prompt_input = gr.Textbox(label="Or enter a prompt", lines=2)

    with gr.Row():
        sdlc_phase = gr.Dropdown(choices=["Requirements", "Design", "Development", "Testing",
"Deployment"], value="Requirements", label="SDLC Phase")

        action = gr.Radio(choices=["Extract Requirements", "Generate Code"], value="Extract
Requirements", label="Action")

    run_button = gr.Button("Run SmartSDLC", elem_classes=["custom-run-button"])

    run_button.click(
        smart_sdlc_run,
        inputs=[file_input, prompt_input, sdlc_phase, action],
        outputs=output_text
    )

demo.launch(debug=True)
```

- **Frontend Development :**
  Design and develop the user interface - using Gradio components like dropdowns, textboxes, and output areas.Create dynamic interaction with the backend - so that user inputs get processed in real-time, and AI outputs (text and image) are displayed seamlessly.

- **Deployment :**
  Prepare the application for local deployment - by launching the Gradio interface using the launch(share=True) or integrating with FastAPI/Streamlit for enterprise use.
  Test and verify - the deployment, ensuring all features work properly and model inference responds as expected. Additional deployment options include Hugging Face Spaces or Docker containers.

These is the project interface:

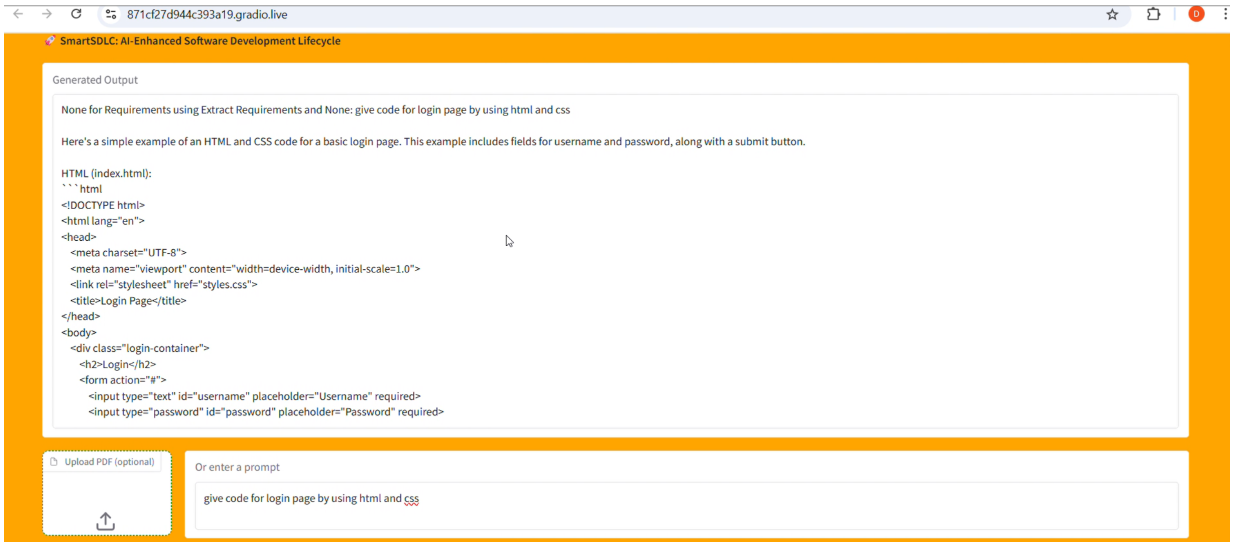


Or enter a prompt: give code for login page by using html and css

SDLC Phase: Requirements

Action: [✓ Extract Requirements] Generate Code

By giving the prompt it give the output as shown in below figure

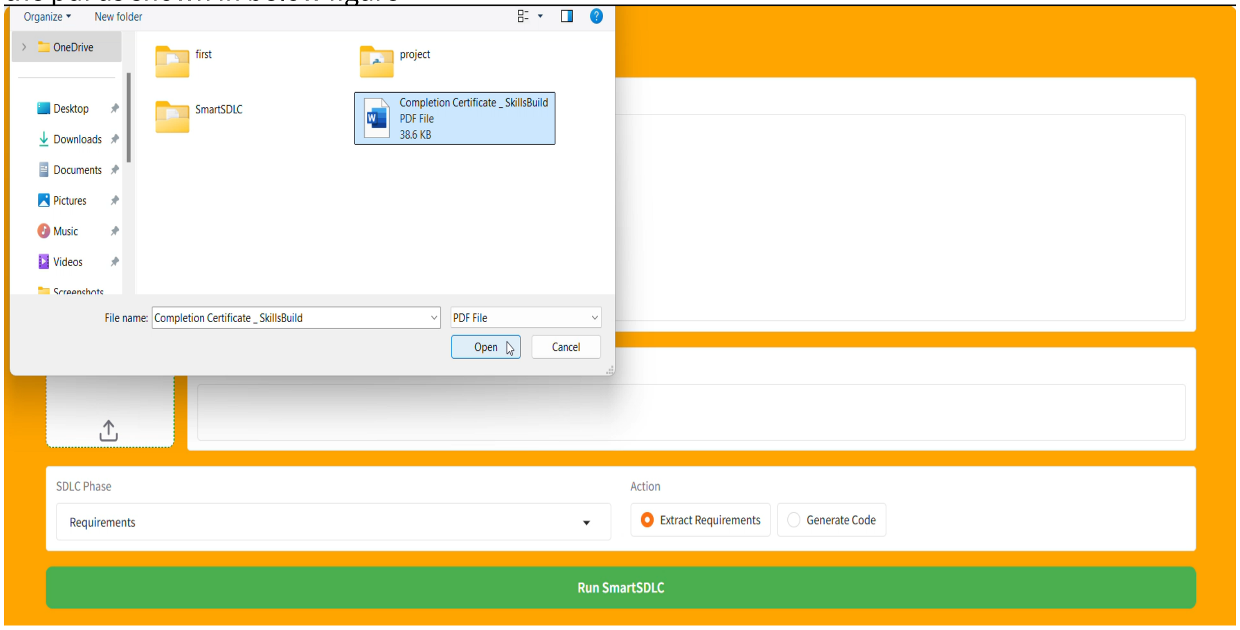


871cf27d944c393a19.gradio.live
SmartSDLC: AI-Enhanced Software Development Lifecycle
Generated Output
Upload PDF (optional)
Or enter a prompt

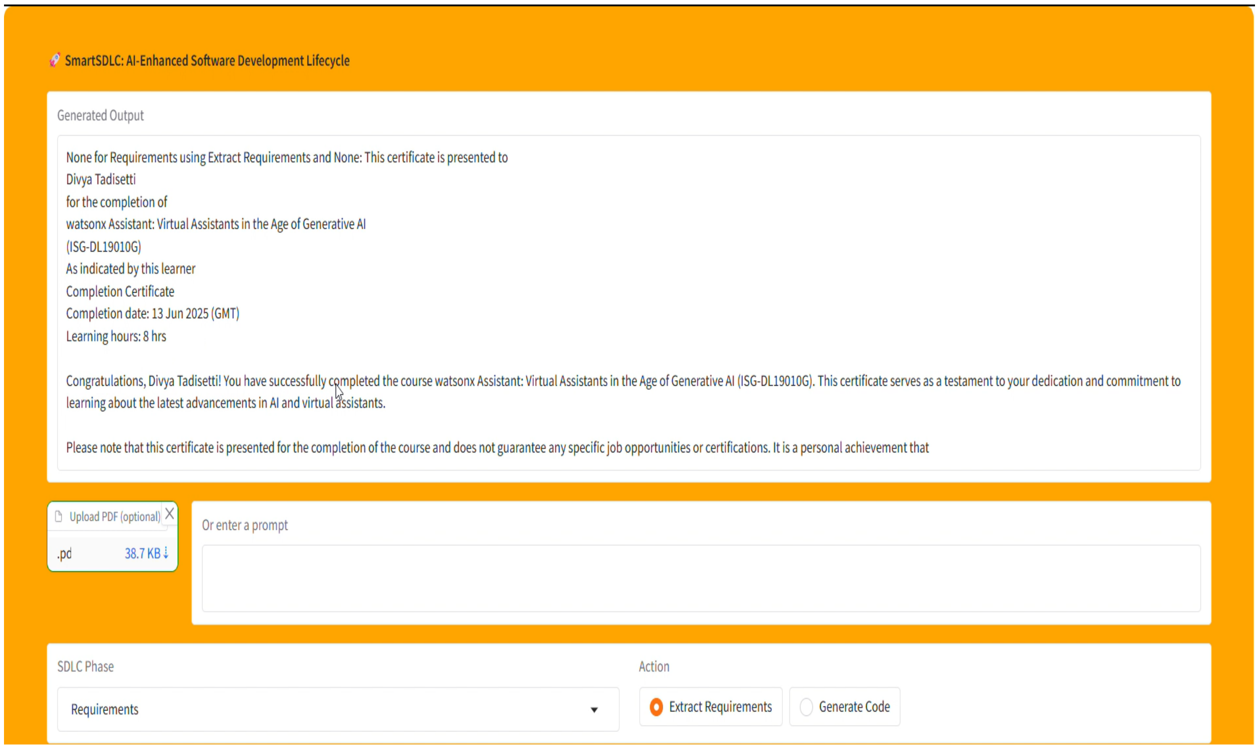
~~~
None for Requirements using Extract Requirements and None: give code for login page by using html and css

Here's a simple example of an HTML and CSS code for a basic login page. This example includes fields for username and password, along with a submit button.

HTML (index.html):
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
  <title>Login Page</title>
</head>
<body>
  <div class="login-container">
    <h2>Login</h2>
    <form action="#">
      <input type="text" id="username" placeholder="Username" required>
      <input type="password" id="password" placeholder="Password" required>
~~~

Or enter a prompt: give code for login page by using html and css

And it has an another functionality is uploading final and analyze the text and extract the text inside the pdf as shown in below figure



And give the output by extract the text inside the uploaded file that shown in below figure


SmartSDLC: AI-Enhanced Software Development Lifecycle

Generated Output

```
None for Requirements using Extract Requirements and None: This certificate is presented to
Divya Tadisetti
for the completion of
watsonx Assistant: Virtual Assistants in the Age of Generative AI
(ISG-DL19010G)
As indicated by this learner
Completion Certificate
Completion date: 13 Jun 2025 (GMT)
Learning hours: 8 hrs

Congratulations, Divya Tadisetti! You have successfully completed the course watsonx Assistant: Virtual Assistants in the Age of Generative AI (ISG-DL19010G). This certificate serves as a testament to your dedication and commitment to learning about the latest advancements in AI and virtual assistants.

Please note that this certificate is presented for the completion of the course and does not guarantee any specific job opportunities or certifications. It is a personal achievement that
```

Upload PDF (optional)
.pd 38.7 KB

Or enter a prompt

SDLC Phase: Requirements

Action: [✓ Extract Requirements] Generate Code

- **Conclusion:**
  Smart SDLC leverages IBM Watsonx and Granite’s generative AI to bring intelligent automation into software development. It enhances productivity across SDLC phases—from Requirements to Maintenance—by offering phase-specific support and generating helpful textual and visual content. The combination of Python, FastAPI, Gradio, and IBM's AI model creates a highly modular and extensible development assistant.

• **Submitted By:**

Team ID:LTVIP2025TMID32013

1) Tadisetti Divya

2) Shaik Mohammed Ilyas

3) Shaik Farooq

4) Shaik Salman