Smart SDLC - AI - Enhanced Software Development Life Cycle

Project Documentation Format

1. Introduction

- **Project Title: Smart SDLC – AI Enhanced Software Development Life Cycle**
- **Team member:**

  Tadisetti Divya : Leader

  Shaik Mohammed Ilyas: Helper

  Shaik Farooq: Helper

  Shaik Salman: Helper

2. Project Overview

SmartSDLC is an AI-powered, full-stack cloud application designed to transform the traditional Software Development Lifecycle (SDLC) by integrating intelligent automation into each core phase. Built on IBM Cloud and powered by the IBM Granite foundation model, SmartSDLC leverages advanced Natural Language Processing (NLP) and Generative AI to convert unstructured inputs—such as raw text requirements or code—into actionable development artifacts like structured user stories, production-ready code, test cases, bug fixes, and documentation. The IBM Granite model serves as the AI engine across the platform, enabling deep understanding and contextual generation of software assets with high accuracy. By automating requirement classification, code generation, debugging, testing, summarization, and real-time support through an integrated chatbot, SmartSDLC significantly enhances productivity, reduces manual effort, and ensures consistency across the SDLC, making it a powerful ecosystem for modern software teams.

★ **Purpose:**

• Automate critical phases of the Software Development Lifecycle (SDLC) using AI.

• Enhance developer productivity by generating code, fixing bugs, and creating test cases automatically.

• Convert unstructured requirements into structured user stories for better clarity and planning.

• Improve software quality and reduce human error with AI-assisted testing and debugging.

• Generate clear, human-readable code summaries to support documentation and onboarding.

• Empower non-technical users to participate in the development process through natural language inputs.

• Provide a unified, intelligent development environment using the IBM Granite foundation model.

• Reduce development time and costs by minimizing manual work and repetitive tasks.

• Leverage IBM Cloud for scalability, reliability, and enterprise-ready deployment.

• Promote innovation by integrating AI and NLP into modern software engineering workflows.

★ Features:

• Upload PDF files containing raw, unstructured requirements

• AI-based classification of content into SDLC phases (Requirement, Design, Development, Testing, Deployment)

• Automatic generation of structured user stories from extracted requirements

• Natural language prompt-to-code generation using IBM Granite model

• AI-powered bug detection and correction for Python and JavaScript code

• Automated test case generation using frameworks like unittest and pytest

• Code summarization with human-readable explanations for documentation and onboarding

• Floating AI chatbot assistant for real-time SDLC guidance and help

• Syntax-highlighted, clean code display on the frontend

• Fully cloud-based deployment on IBM Cloud for scalability and availability

• Modular design — each feature can be used independently or as part of an integrated workflow

• Intelligent prompt routing and keyword detection using LangChain for chatbot responses.

3. Architecture

➤ Frontend:

The frontend of SmartSDLC is built using React.js, providing a responsive and modular interface. It serves as the user-facing layer that handles all interactions, including:

- Uploading requirement PDFs

- Viewing AI-classified SDLC phases
- Displaying AI-generated code, test cases, bug fixes, and summaries
- Providing a syntax-highlighted code viewer
- Hosting a floating AI chatbot assistant for real-time support

➤ **Backend**:

The backend is developed using Node.js with the Express.js framework. It acts as the bridge between the frontend and AI services (powered by IBM Watsonx and IBM Granite model).

➤ **Core Responsibilities:**

- Handle file uploads (PDFs) and extract text using PyMuPDF (via Python child process or microservice)
- Route classification, code generation, bug fixing, and summarization requests to IBM Granite via secure API calls
- Manage authentication (if implemented), sessions, and chatbot prompt routing using LangChain
- Expose RESTful APIs for all frontend operations
- Log and handle user interactions for analytics and feedback

➤ **Database:**

- User profiles and roles (if login/auth is enabled)
- Uploaded requirement documents (metadata, classification results)
- Generated assets: code, test cases, summaries
- Chatbot query history for session continuity
- Audit logs for AI responses (optional for future analytics)

4. Setup Instructions

❖ **Prerequisites**

Before starting the SmartSDLC project in Google Colab, ensure the following are available:

- **Google Account** to access and run Google Colab notebooks
- IBM Cloud Account with access to Watsonx.ai and the Granite 3.2 Instruct model
- IBM Cloud API Key with access rights to Watsonx foundation model
- Colab-compatible Python environment (Colab default: Python 3.10+)
- Required Python packages (listed below)

❖ **Installation & Configuration Steps in Google Colab**

You can run the full project pipeline inside a Google Colab notebook. Here's a step-by-step guide:

1. **Set Up Required Python Packages**

Install required packages in your Colab environment:

!pip install pymupdf

5. Folder Structure

Since SmartSDLC was developed entirely within a Google Colab notebook, the project does not follow a conventional file/folder structure (e.g., separate /client and /server directories). Instead, the entire system is organized into modular notebook cells, each representing logical components of the SDLC automation pipeline.

➢ Notebook-Based Architecture Overview

Instead of folders, the Colab notebook is divided into the following logical sections (cell groups):

1. PDF Upload & Requirement Extraction

- Uses PyMuPDF to read PDF content
- Extracts and preprocesses raw requirement text

2. IBM Granite Model Integration

- Authenticates with Watsonx.ai using the ibm-watson-machine-learning SDK
- Calls Granite v3.2 Instruct for various tasks:
    1. Classifying text into SDLC phases
    2. Generating code
    3. Bug fixing
    4. Test case generation
    5. Summarizing code

3. Classification & User Story Generation

- Splits extracted text into sentences
- Prompts Granite to classify each sentence
- Optionally formats output into structured user stories

4. Code Processing Modules

- Separate cells for:
    1. Code generation from natural language
    2. Bug fixing in Python/JavaScript code
    3. Test case creation
    4. Code summarization

5. Chatbot Assistant (Optional)

- Implements a simple chatbot interface (text input/output)
- Routes user questions about SDLC to Granite with tailored prompts

6. Output Display and Export

- Displays results (code, test cases, summaries) directly in Colab

- Optionally saves outputs to Google Drive or downloads as files

6. Running the Application

The SmartSDLC application is executed within a Google Colab notebook, eliminating the need for local server setup. There are no traditional frontend (npm start) or backend servers involved.

❖ To run the application:

Step 1: Open the Google Colab notebook

Ensure you are logged into your Google account. Open the notebook from Google Drive or GitHub (if hosted).

Step 2: Run all cells in sequence

The notebook is divided into logical sections:

- Upload and extract PDF requirements
- Authenticate and connect to IBM Granite model
- Classify and process SDLC phases
- Generate code, fix bugs, write tests, summarize code
- Interact with the floating chatbot assistant (optional)

Step 3: View outputs directly in the notebook

- Extracted and classified requirements will be displayed in structured format
- AI-generated code, test cases, summaries, and bug fixes are printed below relevant cells
- No additional setup or local server is required

7. API Documentation:

SmartSDLC runs entirely within a Google Colab notebook and directly interacts with IBM Watsonx's Granite v3.2 Instruct model using the IBM Watson Machine Learning Python SDK. It does not expose any public REST API endpoints, but it internally follows a modular, function-based structure for various tasks.

8. Authentication:

- **Hugging Face API Key Authentication**

SmartSDLC uses Hugging Face-hosted IBM Granite v3.2 Instruct model, accessed through

the Hugging Face Inference API. Authentication is handled via a personal API key provided by Hugging Face.

This key allows authorized access to large language models hosted on the Hugging Face

platform without managing a complex user login system.

- **How It Works in the Project:**

You authenticate by passing the API key as a Bearer token in the HTTP request headers:

python

CopyEdit

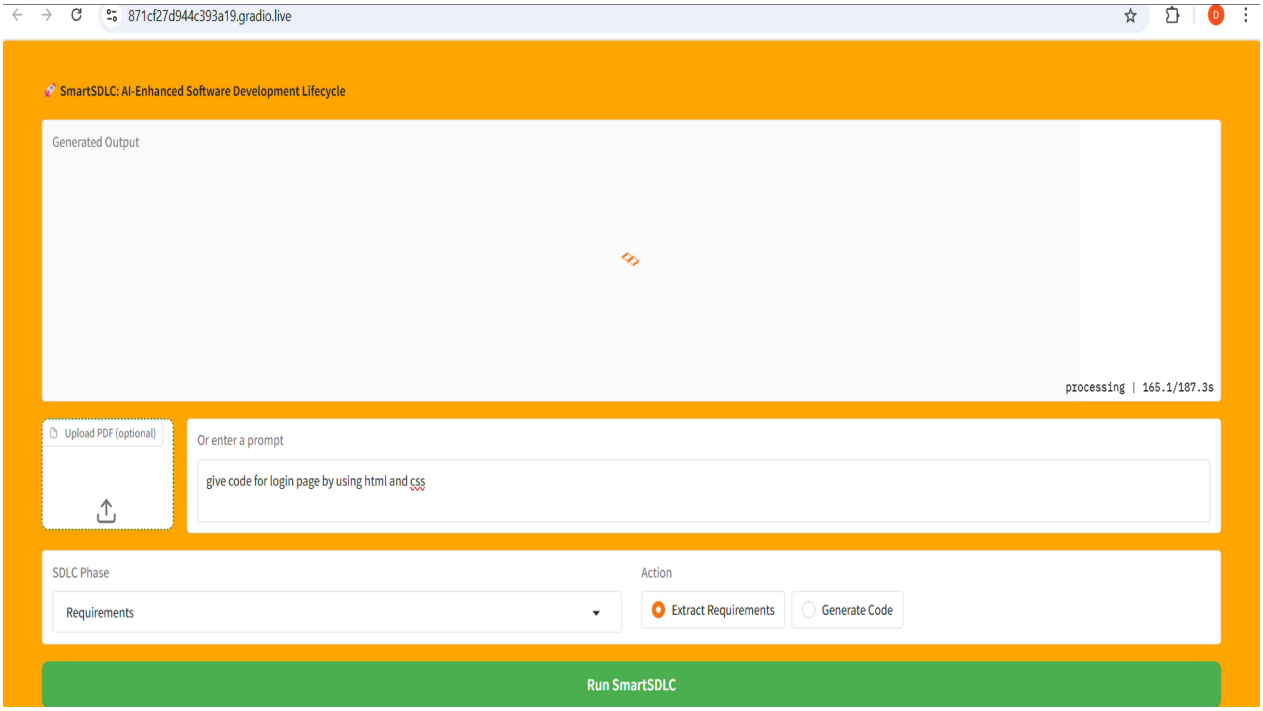
```
import requests
API_URL = "https://api-inference.huggingface.co/models/ibm/granite-3b-instruct"
headers = {
 "Authorization": f"Bearer YOUR_HUGGINGFACE_API_KEY"
}
response = requests.post(API_URL, headers=headers, json={"inputs": prompt})
```

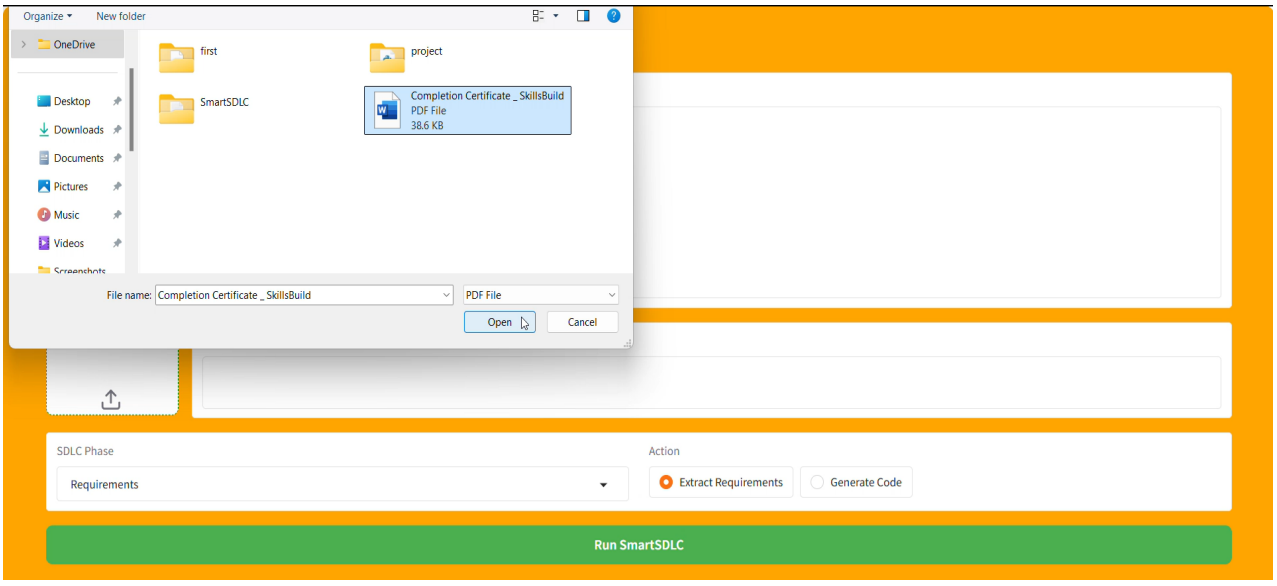
9. User Interface:

SmartSDLC is designed to run in an interactive Google Colab notebook, where each module

functions as a logically separated UI block. Users interact with the system by uploading files,

entering text, and viewing AI-generated outputs directly within the notebook interface.


871cf27d944c393a19.gradio.live
SmartSDLC: AI-Enhanced Software Development Lifecycle
Generated Output
processing | 165.1/187.3s
Upload PDF (optional)
Or enter a prompt
give code for login page by using html and css
SDLC Phase
Requirements
Action
Extract Requirements
Generate Code
Run SmartSDLC


Organize
New folder
OneDrive
Desktop
Downloads
Documents
Pictures
Music
Videos
first
project
SmartSDLC
Completion Certificate _ SkillsBuild
PDF File
38.6 KB
File name: Completion Certificate _ SkillsBuild
PDF File
Open
Cancel
SDLC Phase
Requirements
Action
Extract Requirements
Generate Code
Run SmartSDLC

10. Testing:

➢ Testing Strategy

SmartSDLC-AI is tested through manual testing and functional validation using the Gradio interface. Each module is designed as a user-facing interactive block, allowing rapid iteration and debugging.

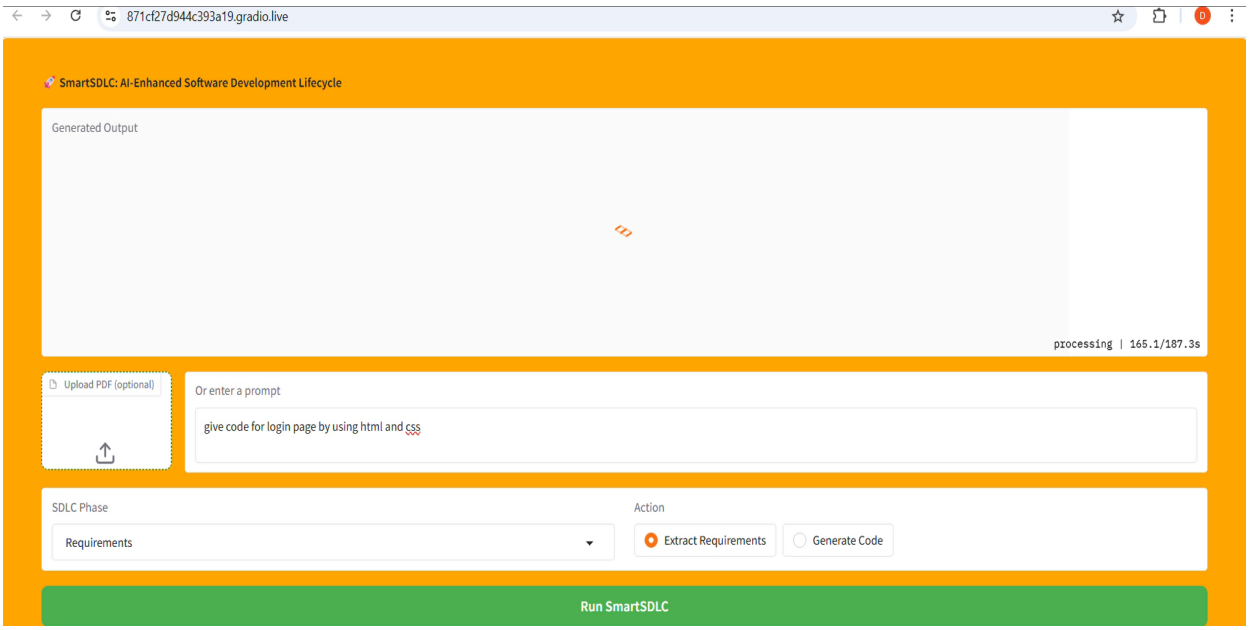
What Was Tested:

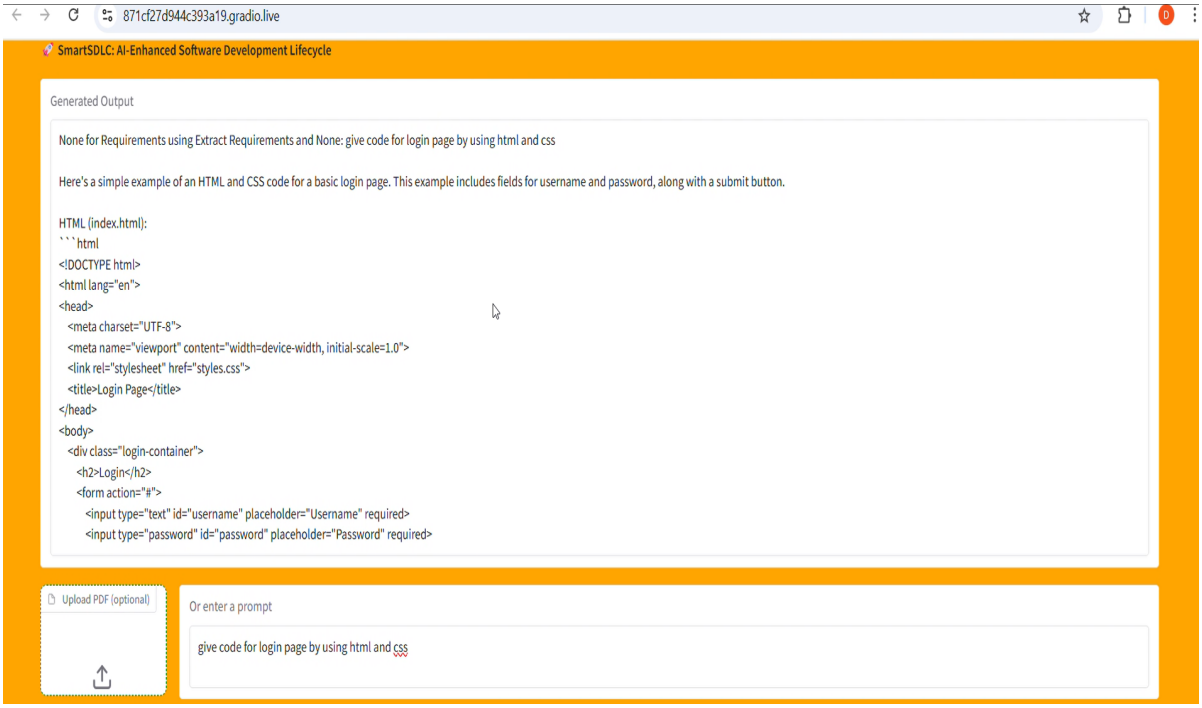
- PDF Parsing: Verified extraction of text from diverse PDF formats using PyPDF2.
- Model Output Validation:
  - Requirement analysis prompts yield structured, relevant functional outputs.
  - Code generation prompts produce working Python code for typical tasks.
- Fallback Model Handling: Ensured system remains operational with a secondary model (DialoGPT-medium) when the IBM Granite model fails to load.

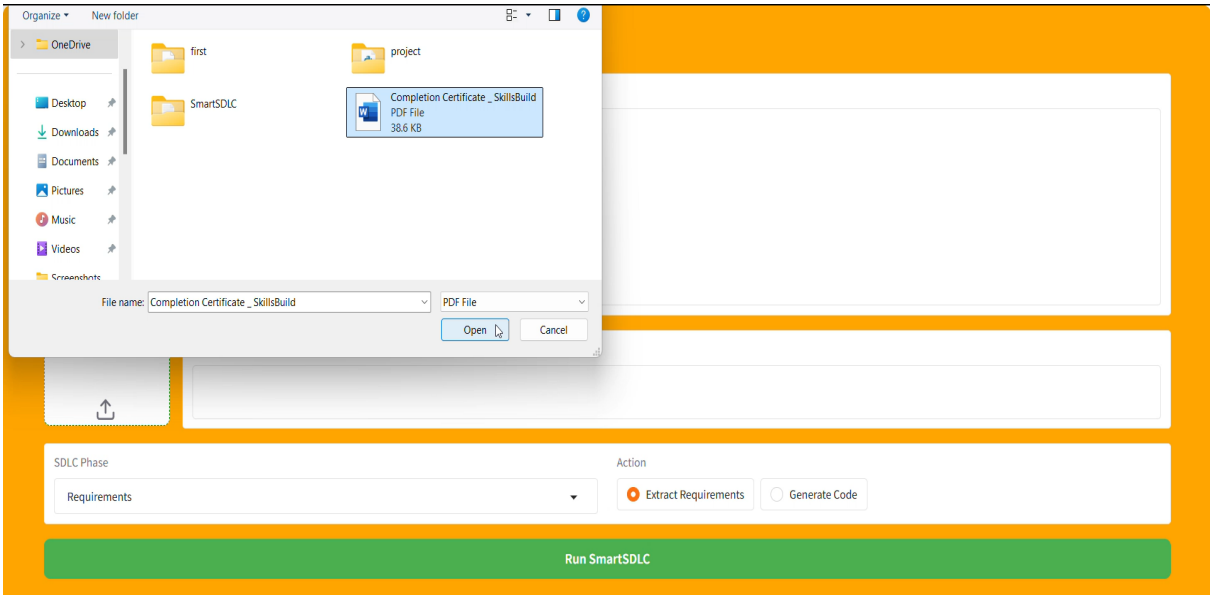
➢ Tools Used:

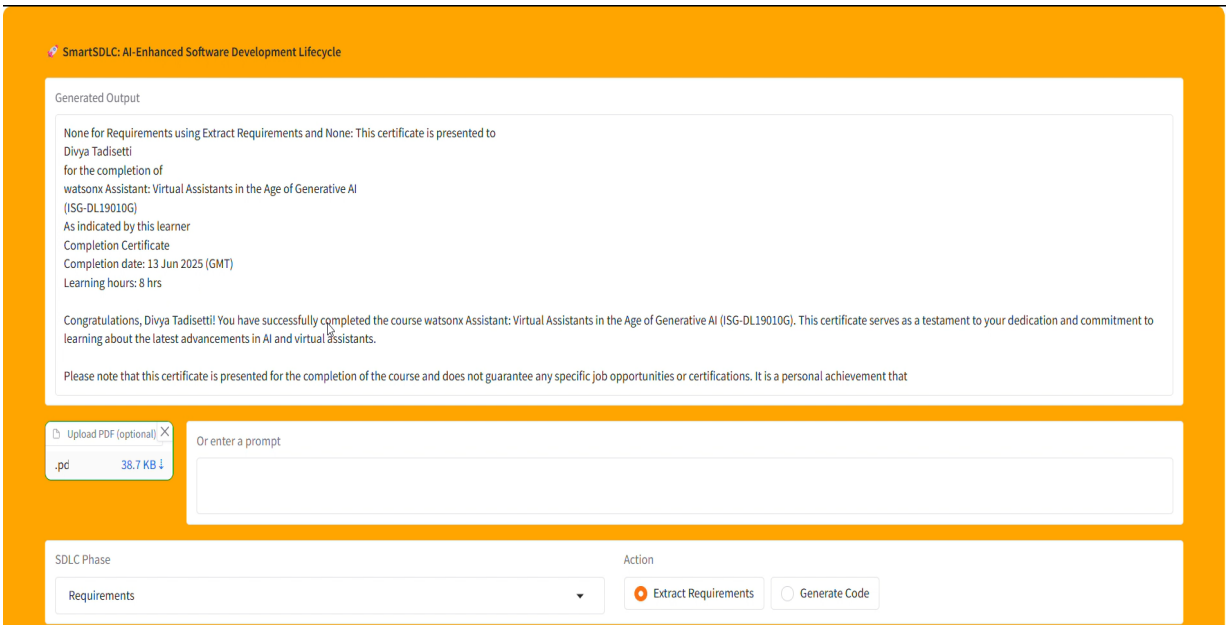
- Google Colab runtime
- Gradio's live UI for visual verification

11. Screenshots or Demo:




871cf27d944c393a19.gradio.live
SmartSDLC: AI-Enhanced Software Development Lifecycle
Generated Output
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>
Upload PDF (optional)
Or enter a prompt
give code for login page by using html and css


Organize
New folder
OneDrive
Desktop
Downloads
Documents
Pictures
Music
Videos
first
project
SmartSDLC
Completion Certificate _ SkillsBuild
PDF File
38.6 KB
File name: Completion Certificate _ SkillsBuild
PDF File
Open
Cancel
SDLC Phase
Requirements
Action
Extract Requirements
Generate Code
Run SmartSDLC
```



12. Known Issues:

Here are some known bugs or limitations that may affect users or developers:

- Model Load Time:

  The IBM Granite 3.3-2B Instruct model is large and can take considerable time to load in Colab, especially on free-tier runtimes.

- Memory Constraints in Google Colab:

  Large models may cause memory overflows or slow performance on limited resources.

- Fallback Model Simplicity:

  The fallback model (DialoGPT-medium) is significantly less capable and may produce

  generic or unrelated outputs.

- PDF Parsing Limitations:

  Some PDFs, especially scanned images or non-standard encodings, may not be parsed accurately by PyPDF2.

- Response Variability:

  Generative AI outputs may vary between runs and may include irrelevant or partially

complete code.

- No Persistent Storage:

  There's no backend database or file-saving mechanism, so all results are lost when the session ends unless manually saved.

- No Authentication:

  The app does not currently restrict access or protect the Hugging Face API key, which can be a security risk in shared environments.

13. Future Enhancements:

To improve usability, scalability, and feature richness, the following enhancements are recommended:

Functional Enhancements:

Add More SDLC Modules:

Incorporate additional tabs for:

- Bug Fixing
- Test Case Generation
- Code Summarization
- Deployment Suggestions

Custom Model Selector:

Let users choose between models (e.g., IBM Granite, GPT-4, Mistral) for different use cases.

Improve Prompt Engineering:

Use structured prompt templates and dropdowns to tailor AI responses more precisely.

Application & UI Improvements:

- Persistent Storage:

Add integration with cloud storage or databases (e.g., Firebase or MongoDB) to save user inputs and outputs.

- Export Functionality:

  Allow users to download generated code and requirement summaries in .txt, .py, or .pdf format.

- Code Execution Cell:

  Let users run generated Python code inside the app using secure sandboxing (e.g.,

  Code Interpreter or subprocess).

- Enhanced Error Handling:

  Provide clearer error messages, especially for model loading, PDF parsing, or API rate limits.

Security Enhancements:

- API Key Protection:

  Store the Hugging Face API key in environment variables or use OAuth-secured proxy to avoid exposure in notebooks.

- User Authentication (Optional):

  Add login functionality if deployed publicly, especially when saving user data.

Deployment Options:

- Dockerize the App:

  Containerize with Docker for easier deployment on IBM Cloud or other cloud platforms.

- Host on a Web Server:

  Convert the app from Colab-based to a fully hosted app using Flask + Gradio, deployed on IBM Cloud or Hugging Face Spaces.