**SmartSDLC**

**Project Documentation**

**Team Members:**

**Kayalvizhi E(Team Leader)**

**Manasha J**

**Jayamshalini S**

**Kalpana M**

# Introduction

The SmartSDLC project is an AI-powered assistant designed to support different stages of the Software Development Life Cycle (SDLC). By integrating natural language models, the project automates requirement analysis, user story generation, and test case creation, while also incorporating gamification to make the process engaging.

# Project Overview

• Purpose: To streamline SDLC processes and assist students and developers in understanding software project stages. • Features: - Requirement Analyzer: Splits raw requirements into functional and non-functional. - User Story Generator: Creates Agile-style user stories. - Test Case Generator: Produces structured test cases. - Gamification: Tracks SDLC Score to motivate consistent use. Policy Summarization & Eco Tips: Additional modules showcasing AI-driven insights.

# Architecture

Frontend: Built with Gradio, providing an intuitive web-based interface. Backend: Uses Hugging Face Transformers and IBM Granite LLM for text generation. Model: Granite 3.3-2B Instruct model powers requirement, policy, and eco analysis. Additional Libraries: PyPDF2 for PDF parsing, Torch for deep learning, Gradio for UI.

# Setup Instructions

Prerequisites: - Python 3.9 or later - pip installed - Required libraries: transformers, torch, gradio, PyPDF2 Installation: - Clone repository - Install dependencies - Run the script to launch the Gradio application

# Source Code

The full source code is included below for reference:

# -\*- coding: utf-8 -\*-

"""Health AI Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1OgXIIUfNZGQOO4ra8MdiC-f9U-iAKcId """ pip install transformers torch gradio PyPDF2 -q

import gradio as gr

import torch from transformers

import Auto Tokenizer, AutoModelForCausalLM import PyPDF2

# load model and tokenizer

model\_name = "ibm-granite/granite-3.3-2binstruct"

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

green\_score = 0

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():

outputs = model.generate(

\*\*inputs,

max\_length=max\_length,

temperature=0.7,

do\_sample=True, pad\_token\_id=tokenizer.eos\_token\_id

)

response = tokenize..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 None

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 eco\_tips\_generator(problem\_keywords):

global green\_score

green\_score += 10 # reward points for generating tips

prompt = f"Generate practical and actionable eco-friendly tips for sustainable living related to:

response = generate\_response(prompt, max\_length=800) return response, f"■ Green Score: {green\_score}"

def policy\_summarization(pdf\_file, policy\_text):

global green\_score

green\_score += 15 # more points for policy analysis if pdf\_file is not None:

content = extract\_text\_from\_pdf(pdf\_file)

summary\_prompt = f"Summarize the following policy document and extract the most important points,key provisions,and implications:\n\n{content}”

else:

summary\_prompt = f"Summarize the following policy document and extract the most important points,key provisions,and implications:\n\n{policy\_text}”

response = generate\_response(summary\_prompt, max\_length=1200) return response, f"■ Green Score: {green\_score}"

# Create Gradio interface with gr.Blocks() as app:

gr.Markdown("# ■ Eco Assistant & Policy Analyzer")

gr.Markdown("Get eco-friendly tips, analyze policies, and grow your \*Green Score\* ■■")

with gr.Tabs():

# Eco Tips Tab

with gr.TabItem("Eco Tips Generator"):

with gr.Row():

with gr.Column():

keywords\_input = gr.Textbox(

label="Environmental Problem/Keywords",

placeholder="e.g., plastic, solar, water waste, energy saving...",

lines=3

)

generate\_tips\_btn = gr.Button("Generate Eco Tips")

with gr.Column():

tips\_output = gr.Textbox(label="Sustainable Living Tips", lines=15)

score\_output = gr.Label(label="Your Green Score")

generate\_tips\_btn.click(

eco\_tips\_generator ,

inputs=keywords\_input,

outputs=[tips\_output, score\_output]

)

# Policy Summarization Tab

with gr.TabItem("Policy Summarization"):

with gr.Row():

with gr.Column():

pdf\_upload = gr.File(label="Upload Policy PDF", file\_types=[".pdf"])

policy\_text\_input = gr.Textbox(

label="Or paste policy text here",

placeholder="Paste policy document text...",

lines=5

)

summarize\_btn = gr.Button("Summarize Policy")

with gr.Column():

summary\_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)

score\_output2 = gr.Label(label="Your Green Score")

summarize\_btn.click(

policy\_summarization,

inputs=[pdf\_upload, policy\_text\_input],

outputs=[summary\_output, score\_output2]

)

app.launch(share=True)

# Future Enhancements

• Add code stub generation in multiple languages (Python, Java, C++). • Integrate UML diagram generation from requirements. • Implement project tracking dashboard for end-to-end SDLC. • Expand gamification with levels like Developer, Architect, and Project Guru. • Enable cloud deployment for team collaboration.

**7.Screenshot**



