SmartSDLC-AI — Enhanced Software Development Lifecycle

Project Documentation

---

1. Introduction

Project Title: SmartSDLC-AI — Enhanced Software Development Lifecycle

Team Members:

- ABINAYA.R
- ADLIN PREETHA.P
- AKSHYA.V
- BHARATHI.R

Purpose:
SmartSDLC-AI is an AI-driven assistant that enhances the Software Development Life Cycle (SDLC) by automating requirement analysis and code generation. It uses an IBM Granite LLM to analyze requirement documents or text and produce categorized requirements (functional, non-functional, and technical) while generating ready-to-use code snippets in multiple programming languages.

---

2. Features

1. **Requirement Analysis**
   - Analyzes PDF documents or text input to extract and categorize software requirements.
   - Implemented via `requirement_analysis()` function using the LLM.
2. **AI Code Generation**
   - Generates working code in multiple programming languages (Python, JavaScript, Java, C++, C#, PHP, Go, Rust).
   - Implemented via `code_generation()` function using the LLM.
3. **PDF Parsing**
   - Extracts text from uploaded PDF files for analysis.
   - Implemented via `extract_text_from_pdf()` using PyPDF2.
4. **Interactive Gradio Interface**
   - Provides a tabbed UI for requirement analysis, code generation, history, and settings.
5. **Authentication**
   - Supports basic login, signup, and logout.

    - Handled via `login()`, `signup()`, `logout()` functions.

6. **History Tracking**
    - Tracks past requirement analyses and code generations per user.
7. **Download Capability**
    - Allows downloading of analyzed requirements and generated code as text files.

---

3. Modules (Code-Aligned)

1. LLM Core

- **Function:** `generate_response(prompt, max_length=1024)`
- **Purpose:** Generates AI responses using IBM Granite LLM.
- **Details:**
    - Tokenizes input.
    - Uses model to generate text.
    - Returns AI-generated response without the original prompt.

---

2. PDF Parser

- **Function:** `extract_text_from_pdf(pdf_file)`
- **Purpose:** Reads uploaded PDF files and extracts text content.
- **Details:**
    - Opens PDF in binary mode.
    - Iterates through pages to extract text.
    - Combines all text and handles errors gracefully.

---

3. Requirement Analyzer

- **Function:** `requirement_analysis(pdf_file, prompt_text, username)`
- **Purpose:** Extracts and organizes software requirements from PDF or text input.
- **Details:**
    - Builds a structured prompt for functional, non-functional, and technical requirements.
    - Calls `generate_response()` to get organized analysis.
    - Saves analysis in `user_history`.

    - Writes analysis to a text file for download.

---

4. Code Generator

- **Function:** `code_generation(prompt, language, username)`
- **Purpose:** Generates code from user-given requirements.
- **Details:**
    - Builds a prompt specifying the programming language.
    - Calls `generate_response()` to produce working code.
    - Saves code in `user_history`.
    - Writes code to a text file for download.

---

5. Authentication

- **Functions:** `login(username, password)`, `signup(username, password)`, `logout()`
- **Purpose:** Manages user login, signup, and logout.

- **Details:**
    - Checks credentials against `USER_CREDENTIALS`.
    - Tracks logged-in users in `current_user`.
    - Updates UI visibility for login/main app.
    - Adds new users and initializes their history.

---

6. History Tracking

- **Implementation:** Inline in Gradio “My History” tab
- **Purpose:** Displays user’s past requirement analyses and generated code.
- **Details:**
    - Uses `user_history` dictionary.
    - Refresh button displays all past analyses and code for the current user.

---

7. Gradio Interface

- **Implementation:** `with gr.Blocks() as app`

- **Purpose:** Provides interactive web UI.
- **Details:**
  - Tabbed layout with:
    - Code Analysis Tab
    - Code Generation Tab
    - History Tab
    - Settings Tab (Logout)
  - Handles file uploads, text input, buttons, outputs, and downloads.
  - Connects all core functions and authentication to UI elements.

---

4. Architecture

Frontend (Gradio)

- Provides a clean, tabbed interface.
- Tabs: Code Analysis, Code Generation, History, Settings.
- Layout built using Blocks, Tabs, Rows, Columns, and Buttons.

Backend (Transformers & PyTorch)

- IBM Granite LLM loaded via Hugging Face Transformers.
- Uses GPU if available, else CPU.

PDF Text Extraction

- Extracts text from uploaded PDF files for requirement analysis.

---

5. Setup Instructions

Prerequisites:

- Python 3.9+
- pip package manager
- Stable internet connection (to download model & dependencies)
- GPU recommended but optional

Installation & Run:

```
git clone
https://github.com/your-username/smartsdlc-ai.git
cd smartsdlc-ai
pip install transformers torch gradio PyPDF2 accelerate
python app.py
```

---

6. Folder Structure

```
smartsdlc-ai/
│
├── app.py                  # Main application
├── requirements.txt        # Dependency list
├── README.md
│
├── assets/
│   └── sample_docs/        # Sample PDF files
```

> Note: All modules and functionality are implemented within `app.py`.

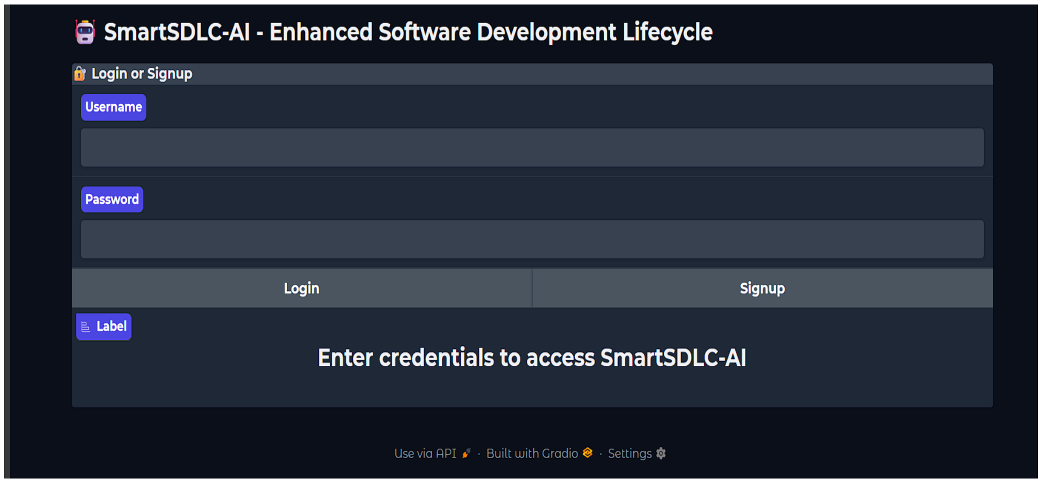
7.Main Code

```
!pip install transformers torch gradio PyPDF2 accelerate -q

# ========================
# 1. Imports
# ========================
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import PyPDF2
import datetime

# ========================
# 2. Load Model (Smaller & Faster)
# ========================
model_name = "google/flan-t5-base"   # faster than Granite 2B
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto"
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# =======================
# 3. Helper Functions
# =======================
def generate_response(prompt, max_length=512):
    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=0.0,      # deterministic, faster
            do_sample=False,      # no random sampling
            pad_token_id=tokenizer.eos_token_id
        )
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return response.strip()

def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""
    try:
        text = ""
        with open(pdf_file.name, "rb") as f:
            pdf_reader = PyPDF2.PdfReader(f)
            for page in pdf_reader.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {str(e)}"

# ======================
# 4. Core Features
# ======================
USER_CREDENTIALS = {"admin": "1234"}  # default user
user_history = {"admin": []}
current_user = {"name": None}

def requirement_analysis(pdf_file, prompt_text, username):
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)[:1500]  # limit text for speed
        prompt = f"Extract functional, non-functional, and technical
requirements:\n\n{content}"
    else:
        prompt = f"Extract functional, non-functional, and technical
requirements:\n\n{prompt_text}"

    result = generate_response(prompt, max_length=512)
    user_history[username].append(("Requirements Analysis", result))

    # Save analysis to a file
    analysis_path = f"analysis_{username}.txt"
    with open(analysis_path, "w", encoding="utf-8") as f:
        f.write(result)

    return result, analysis_path

def code_generation(prompt, language, username):
    prompt_text = f"Generate {language} code for the following
requirement:\n\n{prompt}\n\nCode:"
    result = generate_response(prompt_text, max_length=512)
    user_history[username].append((f"Code ({language})", result))
    # Save code to a file
    code_path = f"code_{username}.txt"
    with open(code_path, "w", encoding="utf-8") as f:
        f.write(result)

    return result, code_path

# ========================
# 5. Login / Signup / Logout
# ========================
def login(username, password):
    if username in USER_CREDENTIALS and USER_CREDENTIALS[username]
== password:
        current_user["name"] = username
        return gr.update(visible=False), gr.update(visible=True), f"✅ Welcome
{username}! Logged in at {datetime.datetime.now().strftime('%H:%M:%S')}"
    else:
        return gr.update(visible=True), gr.update(visible=False), "❌ Invalid
credentials!"

def signup(username, password):
    if username in USER_CREDENTIALS:
        return "⚠️ Username already exists!"
    USER_CREDENTIALS[username] = password
    user_history[username] = []
    return "✅ Signup successful! You can now login."

def logout():
    current_user["name"] = None
    return gr.update(visible=True), gr.update(visible=False), "🔒 Logged out
successfully!"

# ========================
# 6. Gradio Interface
# ========================
with gr.Blocks(theme="soft") as app:
    gr.Markdown("# 🤖 SmartSDLC-AI - Enhanced Software Development
Lifecycle")

    # Login / Signup UI
    with gr.Group(visible=True) as login_ui:
        gr.Markdown("### 🔐 Login or Signup")
        username = gr.Textbox(label="Username")
        password = gr.Textbox(label="Password", type="password")
        with gr.Row():
            login_btn = gr.Button("Login")
            signup_btn = gr.Button("Signup")
        login_status = gr.Label("Enter credentials to access SmartSDLC-AI")

    # Main App (hidden until login)
    with gr.Group(visible=False) as main_ui:
        welcome_text = gr.Markdown("")

        with gr.Tabs():
            # Code Analysis
            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",
lines=5)
                        analyze_btn = gr.Button("Analyze")
                    with gr.Column():
                        analysis_output = gr.Textbox(label="Requirements Analysis",
lines=20)
                        download_analysis = gr.File(label="Download Analysis",
interactive=False)

                analyze_btn.click(
                    requirement_analysis,
                    inputs=[pdf_upload, prompt_input, username],
                    outputs=[analysis_output, download_analysis]
                )
            # Code Generation
            with gr.TabItem("💻 Code Generation"):
                with gr.Row():
                    with gr.Column():
                        code_prompt = gr.Textbox(label="Code Requirements", 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)
                        download_code = gr.File(label="Download Code",
interactive=False)

                generate_btn.click(
                    code_generation,
                    inputs=[code_prompt, language_dropdown, username],
                    outputs=[code_output, download_code]
                )

            # History
            with gr.TabItem("📜 My History"):
                history_output = gr.Textbox(label="Your Past Analyses & Code",
lines=20)
                def show_history(username):
                    return "\n\n".join([f"🔹 {t}: \n{c}" for t, c in user_history.get(username,
[])])
                gr.Button("Refresh History").click(show_history, inputs=username,
outputs=history_output)

            # Settings
            with gr.TabItem("⚙️ Settings"):
                logout_btn = gr.Button("Logout")
            logout_btn.click(logout, outputs=[login_ui, main_ui, login_status])

    # Button actions
    login_btn.click(login, inputs=[username, password], outputs=[login_ui, main_ui,
login_status])
    signup_btn.click(signup, inputs=[username, password], outputs=login_status)

# Launch the app
app.queue().launch(share=True)
```

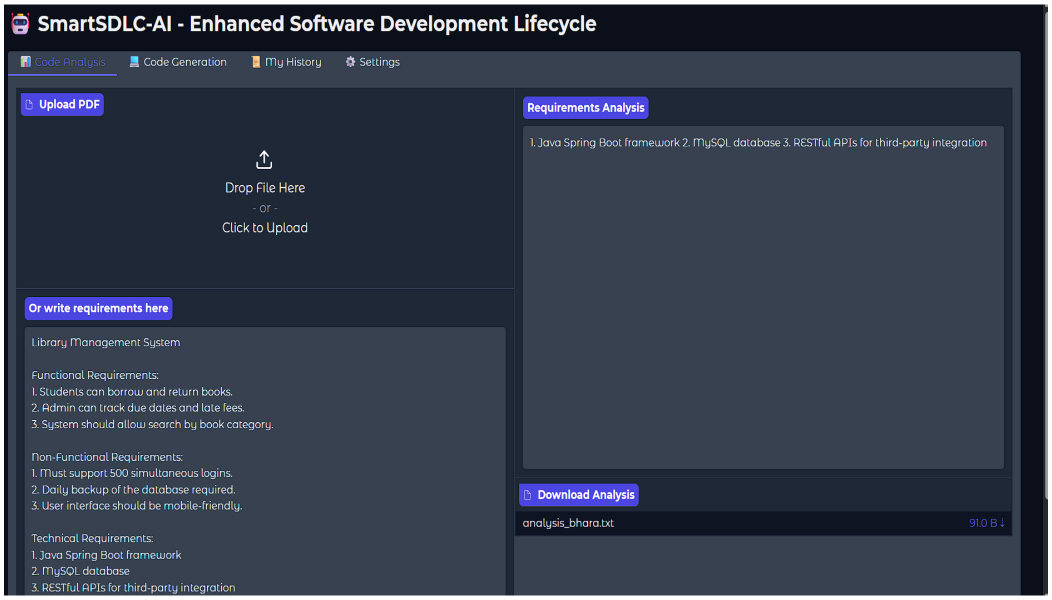
8. Running the Application

1. Launch the Gradio app: `python app.py`
2. Login or Signup
3. **Code Analysis Tab** – Upload PDF or enter text → Click Analyze → View & Download results
4. **Code Generation Tab** – Enter requirement → Select language → Click Generate Code → View & Download results
5. **My History Tab** – View past analyses and code
6. **Settings Tab** – Logout

9.Output


SmartSDLC-AI - Enhanced Software Development Lifecycle
Login or Signup
Username
Password
Login
Signup
Label
Use via API · Built with Gradio · Settings

Enter credentials to access SmartSDLC-AI



Library Management System

Functional Requirements:
1. Students can borrow and return books.
2. Admin can track due dates and late fees.
3. System should allow search by book category.

Non-Functional Requirements:
1. Must support 500 simultaneous logins.
2. Daily backup of the database required.
3. User interface should be mobile-friendly.

Technical Requirements:
1. Java Spring Boot framework
2. MySQL database
3. RESTful APIs for third-party integration

1. Java Spring Boot framework 2. MySQL database 3. RESTful APIs for third-party integration

analysis_bhara.txt 91.0 B

10. Known Issues

- Long PDFs may exceed the model token limit.
- Initial model load may take time.
- Requires internet to fetch model.

11. Future Enhancements

- REST API endpoints
- JWT authentication and user sessions
- Database integration for storing history
- Automatic test case and documentation generation
- Model optimization for faster performance

12. Summary

SmartSDLC-AI streamlines SDLC by combining AI-powered requirement analysis and multilingual code generation within a single web interface. It reduces manual effort, improves quality, and supports rapid prototyping for developers and project teams.