***SmartSDLC: AI-Enhanced Software Development Lifecycle***

**Project overview:**

SmartSDLC leverages IBM's Granite model via Hugging Face to automate and streamline multiple phases of the software development lifecycle. From extracting requirements from PDFs to generating code, writing tests, debugging, and creating documentation, this AI-based system helps developers build high-quality software faster. Deployed on Google Colab using Gradio for an interactive UI, SmartSDLC is ideal for prototyping and educational use.

**🛠️ Technologies Used**

| **Technology** | **Purpose** |
| --- | --- |
| **IBM Granite Model** | NLP model from Hugging Face for code generation, summarization, etc. |
| **Gradio** | Interactive front-end UI in Colab for user interaction |
| **Python** | Core programming language for logic and AI integration |
| **Git & GitHub** | Version control and code hosting |
| **Google Colab (T4 GPU)** | Free GPU-accelerated platform for deploying and testing the app |
| **Hugging Face Transformers** | To access and integrate the IBM Granite models |
| **PyMuPDF / PDFplumber** (optional) | To extract text from uploaded PDFs |

## ****SmartSDLC Project Workflow****

## ****SmartSDLC Project Workflow****

### 🚀 ****Phase 1: Exploration and Planning****

#### ✅ ****Activity 1: Explore Naan Mudhalvan Smart Interz Portal****

* Understand the project expectations and objectives.
* Review example projects or formats shared on the portal.
* Identify deliverables (Colab notebook, GitHub repo, documentation, etc.).

### 📦 ****Phase 2: Model Selection and Setup****

#### ✅ ****Activity 2: Choose an IBM Granite Model from Hugging Face****

* Visit Hugging Face - IBM Granite Models.
* Choose a model suitable for:
  + Code generation
  + Summarization
  + Bug fixing
  + Documentation generation
* Note down the model name (e.g., ibm/granite-13b-instruct) for integration.

### 🛠️ ****Phase 3: Application Development****

#### ✅ ****Activity 3: Develop & Run Application in Google Colab****

##### 🧩 Step-by-Step in Colab:

1. **Set Up Environment**
   * Install necessary libraries:
   * !pip install gradio transformers torch pdfplumber
2. **Build Gradio Interface**
   * Use Gradio to create buttons or text areas for:
     + Upload PDF
     + Generate Code
     + Generate Tests
     + Debug Code
     + Write Docs
     + Chat with AI assistant
3. **Integrate IBM Granite Model**
   * Load model using transformers library:
   * from transformers import pipeline
   * generator = pipeline("text-generation", model="ibm/granite-13b-instruct")
4. **PDF Requirement Extraction**
   * Use pdfplumber or PyMuPDF to extract text from uploaded PDFs.
   * Clean and display extracted requirements.
5. **Prompt Handling**
   * Design prompt templates for:
     + Code generation: “Generate a Python function for user login.”
     + Test generation: “Write unit tests for the given function.”
     + Bug fix: “Fix the following code.”
     + Docs: “Write documentation for the code.”
     + Chat: Free-form questions like “What does this function do?”
6. **Display Outputs in Gradio**
   * Show AI outputs in a code block or text area.

### 📤 ****Phase 4: Deployment and Upload****

#### ✅ ****Activity 4: Upload Project to GitHub****

1. Organize files:
   * SmartSDLC.ipynb
   * README.md
   * Sample PDFs / outputs
   * utils/ (optional for modular code)
2. Initialize Git repo and push to GitHub.

## ****SmartSDLC – Key Features****

### 📄 1. ****PDF Requirements Extraction****

* **Upload a PDF** file (like a client requirement document).
* AI **extracts and summarizes** key software requirements.
* Useful for quickly turning business documents into dev-ready insights.

🛠 Tech used: pdfplumber, IBM Granite model (summarization prompt)

### 💡 2. ****Prompt-to-Code Generation****

* Users can input **natural language prompts** like:

“Create a login page using Flask”  
“Write a REST API for employee management”

* SmartSDLC uses the **IBM Granite model** to generate working code automatically.

🛠 Tech used: Gradio + Hugging Face + IBM Granite

### 🧪 3. ****Test Case Generation****

* Automatically generates **unit tests or test scripts** based on code or prompt.
* Example:

“Write test cases for the login function”

* Great for reducing time spent writing tests manually.

🛠 Tech used: Prompt engineering + code generation model

### 🐞 4. ****Bug Detection and Fixing****

* Input broken or buggy code.
* AI analyzes and **returns a corrected version**.
* Also highlights what the bug was and how it was fixed (if prompted).

🛠 Tech used: Error explanation + code regeneration with AI

### 📚 5. ****Auto Documentation****

* Converts code into **clean documentation** in Markdown or plain text.
* Explains:
  + Functionality
  + Inputs/Outputs
  + Code behavior
* Useful for generating **developer-friendly docs** or user guides.

🛠 Tech used: Prompt: “Explain this code” or “Generate documentation”