

Project Description

*SmartSDLC: AI-
Enhanced
Software
Development
Lifecycle*

SmartSDLC: AI-Enhanced Software Development Lifecycle

TEAM DETAILS:

Team ID: NM2025TMID03731

Team Size: 4

Team Leader: RUBAN D

Team member: JAGATH LAL K K

Team member: RANJETH P

Team member: HARINI V

Introduction

The Software Development Life Cycle (SDLC) is a well-defined, structured methodology used by software developers to systematically design, develop, test, deploy, and maintain software applications. It typically consists of several key stages:


- **Requirement Gathering** – Collecting detailed functional and non-functional requirements from stakeholders.
- **System Design** – Architecting the software structure based on requirements.
- **Implementation** – Writing the actual source code.
- **Testing** – Validating the system to ensure it meets the specified requirements and is free of defects.
- **Deployment** – Releasing the software to production.
- **Maintenance** – Providing ongoing support and updates.

Project Overview

- **Purpose**

The primary purpose of SmartSDLC is to enhance and accelerate the traditional software development lifecycle by integrating intelligent automation tools powered by Artificial Intelligence (AI). In today's software industry, developers often face challenges such as managing vague requirements, repetitive code writing, and ensuring correctness under tight deadlines. SmartSDLC addresses these issues by automating critical phases of the development process, significantly improving efficiency and reducing human error.

SmartSDLC focuses on three main capabilities:

-  **Automated Requirement Extraction:** Requirement documents are typically unstructured, written in natural language, or provided as PDFs or Word files. These often contain ambiguous or incomplete information that requires manual interpretation. SmartSDLC uses advanced Large Language Models (LLMs) to analyze such

documents and extract structured, machine-readable requirements automatically. This eliminates manual transcription work and reduces misinterpretation, accelerating the transition to system design and development.

- ⚡ **Automated Code Generation:** Based on structured requirements, SmartSDLC generates production-ready source code automatically. Developers provide high-level descriptions of desired functionality, and the system produces clean, standardized code following best practices. This removes the burden of writing boilerplate code, ensures consistency, and lets developers focus on complex logic and architecture.
- ✅ **Automated Code Validation and Testing:** After code generation, SmartSDLC performs automated validation to detect syntax errors, logical flaws, and common anti-patterns. It can also generate simple unit tests and verify expected behavior, reducing debugging efforts and increasing code reliability.

Key Benefits

- **Faster Development:** Significant reduction in development time by automating repetitive tasks.
- **Improved Productivity:** Developers focus on creative problem-solving rather than mundane tasks.
- **Consistent Quality:** Standardized code generation and built-in validation minimize errors.
- **Higher Reliability:** Early detection of issues through automated testing improves stability.
- **Scalable Processes:** Structured requirement extraction enables easy scaling across projects.

SmartSDLC empowers development teams to build high-quality software faster, more efficiently, and with fewer errors than traditional approaches.

Architecture

SmartSDLC follows a modular architecture that separates concerns across different layers:

1. Frontend (Gradio)

- Interactive web interface built using Gradio Blocks and Tabs for easy navigation.
- File uploaders, text inputs, dropdown menus, and output displays to manage user interaction.

2. Backend Components

- AI Inference Engine: Uses IBM Granite LLM to process natural language prompts for requirement analysis and code generation.
- PDF Processing Module: PyPDF2 reads PDF files and extracts text into manageable chunks.
- Code Syntax Validator: Uses Python's Abstract Syntax Trees (AST) to parse and validate code syntax.

Sandboxed Execution: Executes generated Python code safely without access to critical system components.

3. Model Storage and Inference

- Pre-trained IBM Granite model loaded via Hugging Face API.
- Utilizes GPU acceleration (if available) for faster inference.

Setup Instructions

- Clone the project repository or download source files.
- Install required dependencies:
- `pip install transformers torch gradio PyPDF2`
- Ensure you have access to the IBM Granite model either from Hugging Face or IBM's official repository.
- Open the main notebook file (SmartSDLC.ipynb) in Google Colab, Jupyter Notebook, or your local Python IDE.
- Execute the notebook cells sequentially to initialize the environment, load the model, and launch the Gradio interface.

Folder Structure

SmartSDLC/

|

└─ SmartSDLC.ipynb # Main application logic, including UI flow, model inference, and interaction handling

└─ requirements.txt # List of all required Python libraries and dependencies, used for easy environment setup

|

└─ assets/ # Contains custom CSS files, static images, and any additional frontend assets used for styling the app

|

└─ models/ # Directory for pre-trained AI models if stored locally, allowing offline or custom model usage

|

└─ tests/ # Unit tests and test scripts designed to validate the correct behavior of individual modules and system workflows

|

|└─ docs/ # Comprehensive project documentation, including design manuals, user guides, and technical references

|


|└─ README.md # High-level project overview, setup instructions, usage examples, and contribution guidelines


|

|└─ config/ # Configuration files for future enhancements such as environment variables, model parameters, or deployment settings

Running the Application

- Open the project in Google Colab, Jupyter Notebook, or local Python environment.
- Run the entire notebook.
- The Gradio application will start and provide a web URL for interaction.
- Navigate between the two main tabs:

1.  Code Analysis: Upload a PDF or type requirement descriptions to extract structured requirement specifications.

2.  Code Generation: Provide high-level code requirements, select a programming language, and generate runnable code.

- Generated outputs will be displayed instantly in the interface.


API Documentation

- `generate_response(prompt, max_length=1024)`:Generates a deterministic response from the language model based on the given prompt.
- `extract_text_chunks(pdf_file, chunk_size=2000)`:Extracts text from PDF and splits it into chunks of specified size to handle large documents.
- `validate_python_code(code)`:Performs syntax validation using Python's `ast.parse()` and returns a success flag with a descriptive message.
- `test_python_code_execution(code)`:Executes Python code in a restricted environment to detect runtime errors and prevent unsafe code execution.
- `requirement_analysis(pdf_file, prompt_text)`:Analyzes the provided PDF or manual requirement input and returns structured output in
- `code_generation(prompt, language)`:Generates runnable code in the selected language, performs syntax validation (for Python), and executes simple tests.

Authentication


- At present, the application does not support authentication mechanisms.
- For production environments, future versions will implement:
 - OAuth 2.0 integration
 - API Key-based user validation
 - Role-based access control (Admin, Developer, etc.)

User Interface

- The interface is designed with usability and aesthetics in mind:
-  Gradient dark background for a modern look.
- Tabs separate Requirement Analysis and Code Generation workflows.
- File upload support for PDFs with drag & drop functionality.
- Text inputs with placeholder guidance.
- Dropdown menu for selecting supported programming languages.
- Action buttons to trigger analysis or code generation.
- Output textboxes with scrollable content for results.

- Responsive layout with rounded corners and soft shadows for visual appeal.

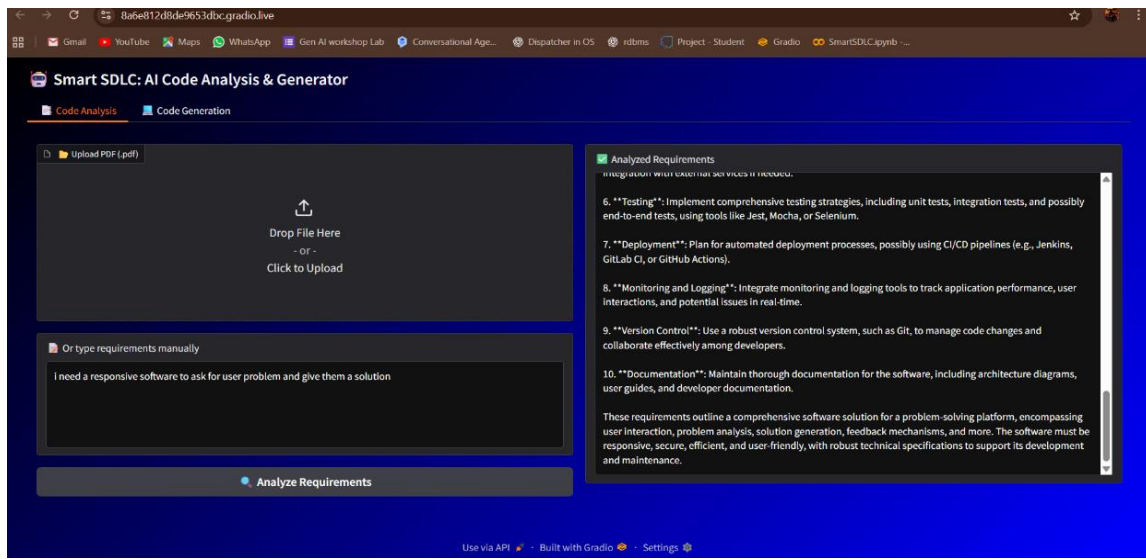
Test Cases

- Upload valid PDF -Extract structured text without errors
- Upload corrupted PDF-Return error message " Error reading PDF"
- Manual input for requirements-Should analyze and output functional/non-functional requirements
- Generate valid Python code-Output code with no syntax errors and successful execution message
- Generate invalid Python code-Mark syntax error and display error details
- Test sandboxed code execution-Should not access unauthorized system functions

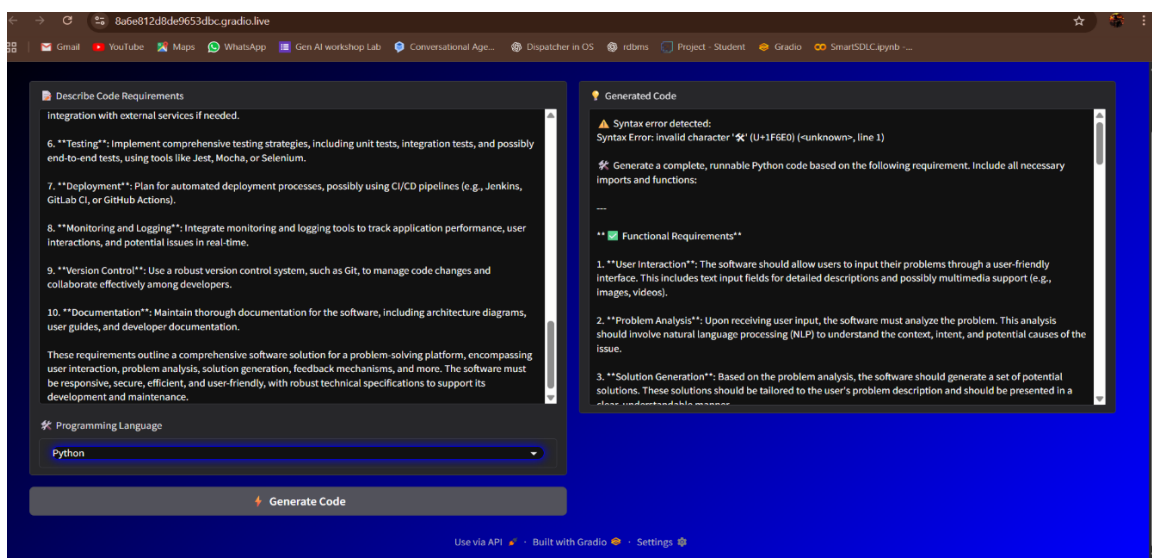
These tests ensure the application behaves reliably under a variety of conditions.

Project Output






1. Give a Prompt in the code analysis section and click on Analyze Requirement. It will Generate The Descriptive Requirement Analysis Automatically.







2. Give the Project Requirements in the code Generation Section. It will Generate the software code as per the given instructions.



Known Issues

-  Large PDF documents may be only partially processed due to chunk size limits.
-  The sandboxed code execution environment is minimal and may not catch all dangerous code patterns.
-  Model loading time can be slow in CPU-only environments.
-  No persistent data storage or multi-user functionality is currently implemented.
-  No logging mechanism to track user activities or errors.

Future Enhancements

-  Implement OAuth or API key-based authentication for secure access.
-  Add a database backend to store user requirements, generated code, and history logs.
-  Implement advanced PDF parsing that handles multi-page tables, images, and complex formatting.
-  Optimize inference speed by enabling model quantization and caching techniques.

- 📄 Add support for multi-language syntax validation and code execution environments.
- ☁ Deploy as a fully hosted SaaS with scalable APIs for external integrations.
- ☐ Enable collaboration features like version control of requirements and code snippets.

Conclusion

SmartSDLC: AI-Enhanced Software Development Lifecycle streamlines software development by automating key processes such as requirement extraction, code generation, and validation using advanced AI models. This reduces manual effort, speeds up development, and improves code quality. By enabling developers to focus on complex tasks, SmartSDLC enhances productivity and ensures more reliable, scalable software solutions, paving the way for efficient, future-ready development workflows.

As technology continues to evolve, SmartSDLC offers a flexible and extensible framework that can adapt to new challenges and emerging trends. Its intelligent automation capabilities provide a strong foundation for integrating further innovations, such as advanced testing, multi-language support, and continuous

integration, making it an ideal solution for modern software development teams aiming for agility and efficiency.