Aditya College Of Engineering & Technology
Department of Artificial intelligence & Machine Learning

INTERNSHIP REPORT

ON

“Smart sdlc-Ai-Enhanced software development life cycle”

Submitted in partial fulfillment of the requirements of the

Virtual Internship Program

Organized by

SMART INTERNZ

Submitted by

Sandeep Jakka

K Sudeep

Kopparthy Varun

Kokkigedda Sameer Nandan

TEAM ID:
LTVIP2025TMID29828

Under the Mentorship of

Mr. Ganesh sir

Smart Bridge

June 2025

Index

S. No.	Section Title	Page No.
1.	INTRODUCTION	1
2.	IDEATION PHASE	1
3.	REQUIREMENT ANALYSIS	2-3
4.	PROJECT DESIGN	3
5.	PROJECT PLANNING & SCHEDULING	4
6.	FUNCTIONAL & PERFORMANCE TESTING	4
7.	RESULT	4-5
8.	ADVANTAGES & DISADVANTAGES	6
9.	CONCLUSION	6
10.	FUTURE SCOPE	6
11.	APPENDIX	6

1. INTRODUCTION

1.1 Project Overview

SmartSDLC is a full-stack AI-powered application designed to automate and enhance the software development life cycle (SDLC). By using advanced NLP and AI models like IBM Granite, SmartSDLC enables users to interact with a chatbot to generate documents, write code, create test cases, and track feedback—all in natural language.

1.2 Purpose

The purpose of this project is to reduce the manual effort involved in software development and to improve productivity and communication across development teams by using AI to handle repetitive and time-consuming SDLC tasks.

2. IDEATION PHASE

2.1 Problem Statement

Traditional SDLC processes are manual, repetitive, and error-prone. Delays in documentation, miscommunication between teams, and inconsistent code/test generation create bottlenecks. A smart solution is required to automate and streamline these processes.

2.2 Empathy Map Canvas

- **Users**: Developers, Testers, Project Managers, Students
- **Needs**: Faster development, less documentation work, code/test automation
- **Pains**: Time delays, unclear requirements, repetitive tasks
- **Gains**: Increased efficiency, error reduction, learning support

2.3 Brainstorming

During the brainstorming sessions, we identified key modules to be automated: requirement gathering, SRS generation, design diagrams, code boilerplate creation, test case automation, and sentiment-based feedback analysis. IBM Granite was selected for its capability to handle NLP tasks efficiently.

3. REQUIREMENT ANALYSIS

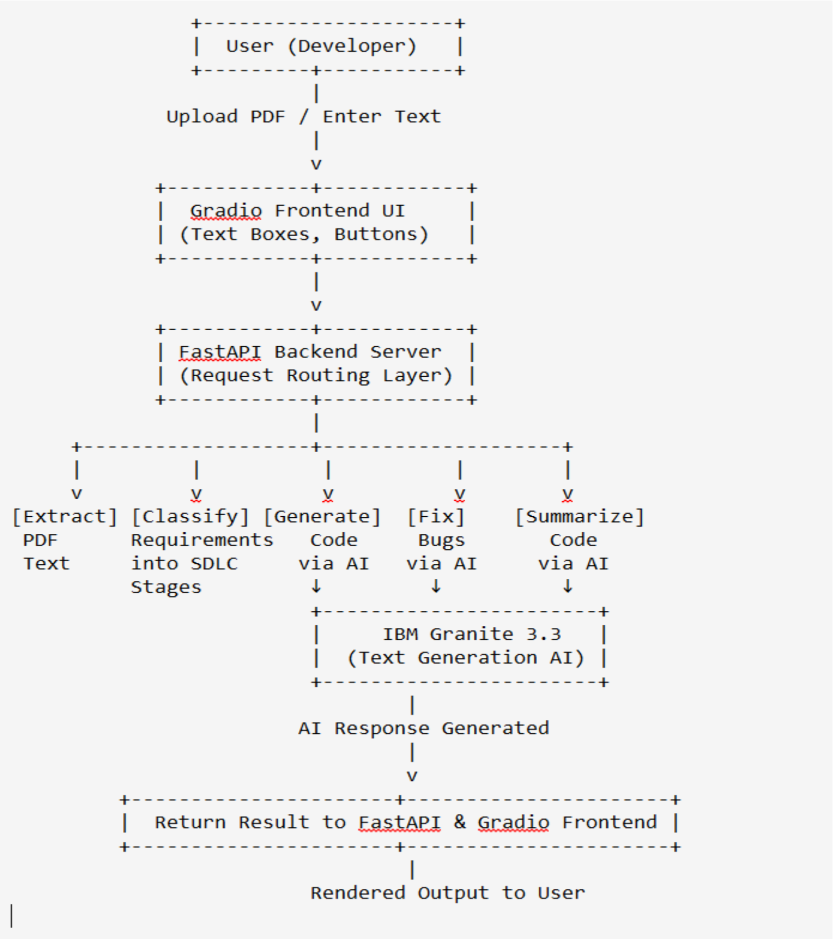
3.1 Customer Journey Map

1. **Start**: User logs into the SmartSDLC interface
2. **Middle**: User provides project requirements via chatbot
3. **End**: Receives SRS, code, test cases, and feedback dashboard

3.2 Solution Requirements

- AI assistant chatbot
- Natural language input processing
- Auto document generation (SRS)
- Auto code and test generation
- Sentiment analysis on feedback
- Dashboard visualization

3.3 Data Flow Diagram



```
                    +---------------------+
                    |  User (Developer)   |
                    +----------+----------+
                               |
                 Upload PDF / Enter Text
                               |
                               v
                 +-------------+-----------+
                 |  Gradio Frontend UI     |
                 | (Text Boxes, Buttons)   |
                 +-------------+-----------+
                               |
                               v
                 +-------------+-----------+
                 | FastAPI Backend Server  |
                 | (Request Routing Layer) |
                 +-------------+-----------+
                               |
       +-----------------------+---------------------+
       |          |          |          |          |
       v          v          v          v          v
[Extract] [Classify] [Generate]  [Fix]    [Summarize]
 PDF      Requirements   Code      Bugs       Code
 Text     into SDLC     via AI    via AI     via AI
          Stages           ↓         ↓          ↓
                       +----------------------+
                       |    IBM Granite 3.3   |
                       |  (Text Generation AI)|
                       +----------------------+
                                |
                       AI Response Generated
                                |
                                v
         +----------------------+----------------------+
         |  Return Result to FastAPI & Gradio Frontend |
         +----------------------+----------------------+
                                |
                        Rendered Output to User
|
```

3.4 Technology Stack

- **Frontend**: HTML, CSS, JavaScript (Bootstrap)
- **Backend**: Python, Flask
- **AI**: IBM Granite / Hugging Face Transformers / granite-3.3-2b-instruct
- **Database**: SQLite/PostgreSQL
- **Tools**: Git, Postman, VS Code, Google Colab

4. PROJECT DESIGN

4.1 Problem Solution Fit

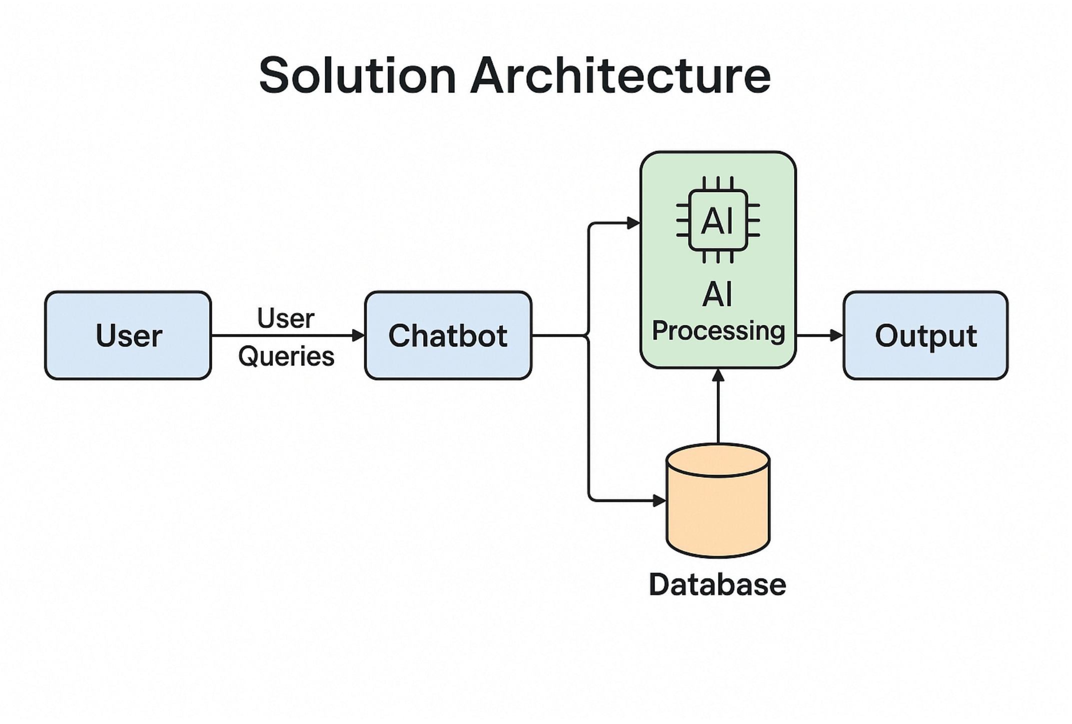
SmartSDLC targets inefficiencies in SDLC by introducing AI-powered tools that produce fast, reliable, and structured outputs based on plain-text inputs.

4.2 Proposed Solution

A modular AI application where the user can interact with a chatbot to automate:

- Requirements classification
- Design diagram generation
- Code and test generation
- Feedback analysis

4.3 Solution Architecture



5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

Week	Task
1	Flask backend setup and IBM Granite integration
2	Chatbot development and feedback collection module
3	Dashboard creation and sentiment integration
4	Final testing, deployment, and documentation

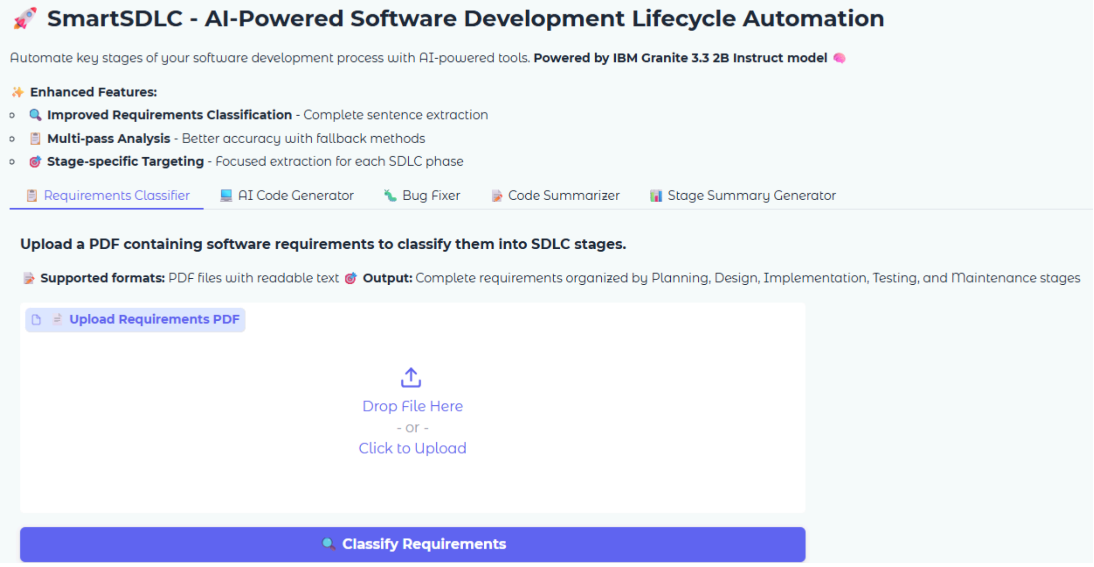
6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

Test cases included chatbot accuracy, latency (under 3 seconds), SRS document format verification, test case logic, and UI responsiveness. Tools like Postman and Pytest were used to ensure module-wise and integrated functionality.

7. RESULTS

7.1 Output Screenshots



SmartSDLC - AI-Powered Software Development Lifecycle Automation

Automate key stages of your software development process with AI-powered tools. **Powered by IBM Granite 3.3 2B Instruct model**

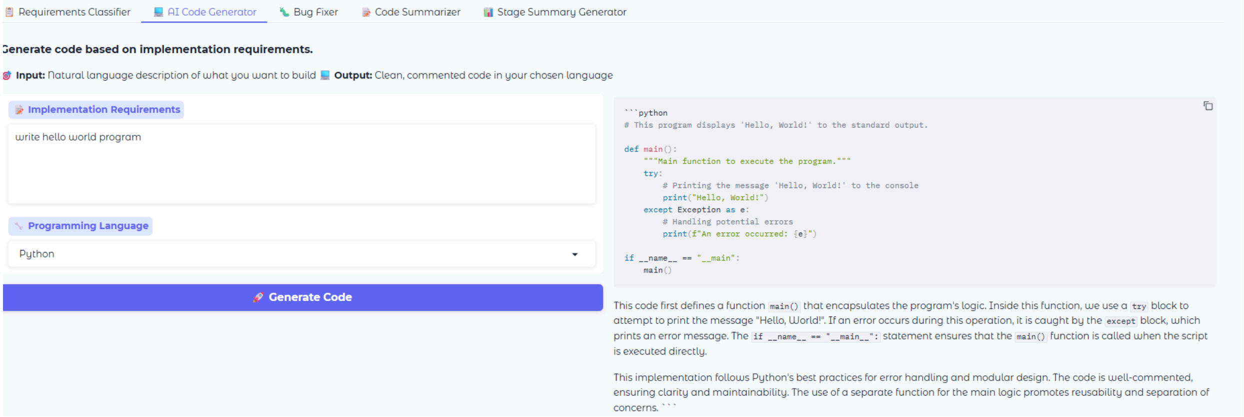
Enhanced Features:

- **Improved Requirements Classification** - Complete sentence extraction
- **Multi-pass Analysis** - Better accuracy with fallback methods
- **Stage-specific Targeting** - Focused extraction for each SDLC phase

Upload a PDF containing software requirements to classify them into SDLC stages.

Supported formats: PDF files with readable text **Output:** Complete requirements organized by Planning, Design, Implementation, Testing, and Maintenance stages

(i) pdf requirement classifier



Generate code based on implementation requirements.

Input: Natural language description of what you want to build **Output:** Clean, commented code in your chosen language

write hello world program

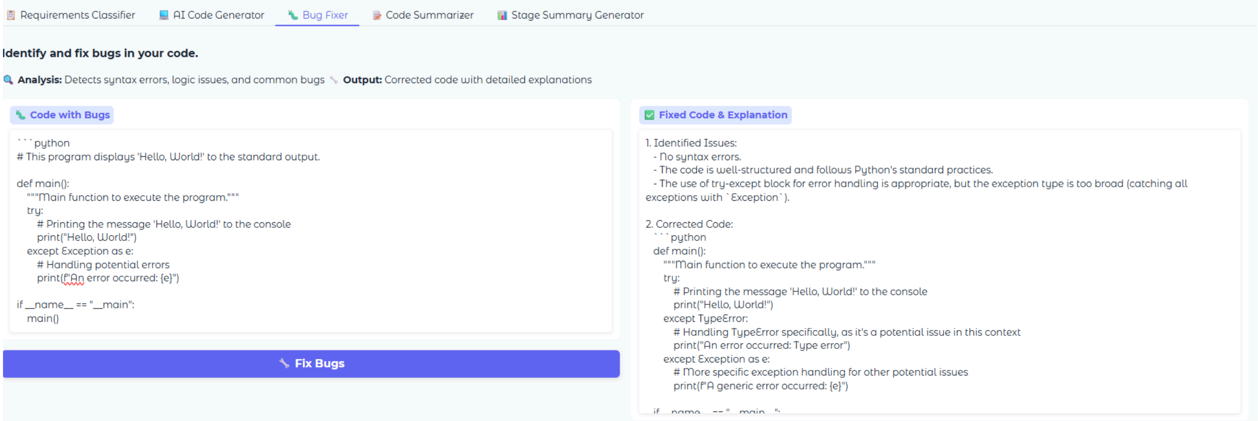
Python

~~~
```python
# This program displays 'Hello, World!' to the standard output.

def main():
    """Main function to execute the program."""
    try:
        # Printing the message 'Hello, World!' to the console
        print("Hello, World!")
    except Exception as e:
        # Handling potential errors
        print(f"An error occurred: {e}")

if __name__ == "__main":
    main()
~~~

This code first defines a function `main()` that encapsulates the program's logic. Inside this function, we use a `try` block to attempt to print the message "Hello, World!". If an error occurs during this operation, it is caught by the `except` block, which prints an error message. The `if __name__ == "__main__":` statement ensures that the `main()` function is called when the script is executed directly.

This implementation follows Python's best practices for error handling and modular design. The code is well-commented, ensuring clarity and maintainability. The use of a separate function for the main logic promotes reusability and separation of concerns. ```

(ii) Code Generator



Identify and fix bugs in your code.

Analysis: Detects syntax errors, logic issues, and common bugs **Output:** Corrected code with detailed explanations

~~~
```python
# This program displays 'Hello, World!' to the standard output.

def main():
    """Main function to execute the program."""
    try:
        # Printing the message 'Hello, World!' to the console
        print("Hello, World!")
    except Exception as e:
        # Handling potential errors
        print(f"An error occurred: {e}")

if __name__ == "__main":
    main()
~~~

~~~
1. Identified Issues:
  - No syntax errors.
  - The code is well-structured and follows Python's standard practices.
  - The use of try-except block for error handling is appropriate, but the exception type is too broad (catching all exceptions with `Exception`).

2. Corrected Code:
  ```python
  def main():
     """Main function to execute the program."""
     try:
        # Printing the message 'Hello, World!' to the console
        print("Hello, World!")
     except TypeError:
        # Handling TypeError specifically, as it's a potential issue in this context
        print("An error occurred: Type error")
     except Exception as e:
        # More specific exception handling for other potential issues
        print(f"A generic error occurred: {e}")

  if __name__ == "__main__":
~~~

(iii) Bug Fixer

8. ADVANTAGES & DISADVANTAGES

Advantages:

- Reduces development time and effort
- Generates structured and consistent outputs
- Simplifies complex SDLC tasks using plain English
- Assists learners and small teams

Disadvantages:

- May generate incorrect results with vague input
- API latency dependent on network and model load
- No authentication in prototype phase

9. CONCLUSION

SmartSDLC simplifies software development by intelligently automating crucial phases using natural language. It allows developers and learners to generate documents, code, and tests quickly, making SDLC more accessible and efficient.

10. FUTURE SCOPE

- Add authentication and role-based access
- Introduce bug fixing and code summarizer
- GitHub and deployment integration
- Voice input support and multi-language chatbot
- Expand UI/UX with advanced analytics and CI/CD tools

11. APPENDIX

- **Source Code**: https://github.com/SandeepJakka/SmartSDLC/
- **GitHub & Project Demo**: https://github.com/SandeepJakka/SmartSDLC/