SMART BRIDGE INTERNSHIP

PROJECT TITLE:

**Smartsdlc** —

Ai-Enhanced software Development Lifecycle .

Submitted by:

Team Leader:-Shaik Mehrudhin (23481a04k3)

Team Member:-Sanaka Nandini Sriya (23481A12K2)

Team Member:-Siddineni Reethika Sri(23481a04k8)

**1.INTRODUCTION**

**1.1 PROJECT OVERVIEW:-**

SmartSDLC is a Generative AI–powered application designed to assist and automate various phases of the Software Development Lifecycle (SDLC). It leverages advanced AI models—such as those from Hugging Face (IBM Cloud Instruct) and IBM WatsonX—to accelerate software engineering tasks, reduce manual effort, and improve overall software quality.

**1.2 PURPOSE:-**

HIGHLIGHT KEY FEATURES AND FUNCTIONALITIES:

1. Document Analysis:

Extracts and classifies requirements from uploaded PDFs or documents using NLP.

2. Code Generation:

Generates Python code based on natural language requirements.

3. Bug Fixing:

Automatically detects and suggests fixes for buggy code.

4. Code Documentation:

Summarizes and explains code to generate human-readable documentation.

5. Chatbot Support:

Offers an AI assistant interface to help developers interact conversationally with the system.

**2.IDEATION PHASE**

**2.1 PROBLEM STATEMENT**:

Manual SDLC processes are time-consuming, error-prone, and inefficient. Developers struggle with:

* Extracting and understanding requirements
* Writing and testing code from scratch
* Fixing bugs and maintaining documentation

SmartSDLC solves this by automating tasks with Generative AI — improving speed, accuracy, and developer productivity.

**2.2 EMPATHY MAP CANVAS**:

Target User:

Software Developers, QA Engineers, Team Leads.

Says:

“Fixing bugs delays our release deadlines.”

“I wish there was a tool that could help me code faster.”

Does:

Writes and tests code ,Uploads documents for requirement analysis,Searches online for bug fixes.

Feels:

Frustrated by repetitive tasks,Overwhelmed with deadlines,Cautiously curious about AI tools

Relieved when tasks are automated successfully.

**2.3 BRAINSTROMING**:

* **Planning**

Use **Watson NLP & Discovery** to analyze user needs.

GenAI creates user stories & estimates timelines.

* **Requirement Analysis**

Watson detects ambiguities, contradictions in requirements.

AI suggests edge cases and missing validations.

* **Design**

GenAI generates architecture diagrams, class models.

IBM Knowledge Catalog recommends design patterns.

* **Implementation**

**Watson Code Assistant** helps with code generation & reviews.

AI suggests secure code practices and documents code logic.

* **Testing**

AI generates test cases from requirements.

Watson AIOps predicts risky code areas and bugs.

* **Deployment**

SmarTSDLCI/CD with failure prediction.

AI-generated changelogs and rollback decisions.

* **Maintenance**

Watson AIOps automates monitoring and ticket prioritization.

Predicts future failures and suggests proactive fixes.

During our brainstorming session, we explored how to integrate AI into each stage of the SDLC using IBM Cloud services. Below is a summary of ideas generated for Planning, Design, Development, Testing, Deployment, and Maintenance phases, highlighting the role of Watson NLP, GenAI SDK, and Watson AIOps.

**3. REQUIREMENT ANALYSIS**

**3.1 Customer Journey map:**

1. Discover

↓

2. Onboard

↓

3. Plan

↓

4. Design

↓

5. Develop

↓

6. Test

↓

7. Deploy

↓

8. Monitor

**3.2 Solution Requirement**

1. Purpose

Enable automatic, context-aware generation of comprehensive documentation for software modules, systems, and API endpoints using generative AI.

2. Key Functionalities

Contextual Summary Generation: Produce human-readable summaries of code modules, algorithms, and workflows.

Inline Code Commentary: Insert meaningful inline documentation within code files.

API Doc Generation: Auto-generate OpenAPI-style documentation for FastAPI routes.

3. User Interactions

Upload or paste code and configuration files.

Review and edit generated content before exporting or pushing to repositories.

4. Technology Stack

Model Integration: IBM Cloud Instruct models via Hugging Face transformers.

Backend: FastAPI to orchestrate processing requests.

Frontend: HTML + CSS with interactive doc preview.

Hosting: Deployed on IBM Cloud, with integration into Google Colab notebooks for developer testing.

5. Output Formats

Markdown (.md)

HTML pages

PDF (via LaTeX export)

6. Edge Cases Handled

Duplicate comments detection

Outdated doc auto-refresh flagging

Language preference (multilingual support)

**3.3 Data Flow Diagram**

A diagram of a software development

AI-generated content may be incorrect.

**3.4 Technology Stack:**

**Frontend**

**Backend**

**4. PROJECT DESIGN**

**4.1 Problem Solution Fit**

**Problem 1:** Manual code documentation is time-consuming and often outdated.

**Solution**: Auto-generate context-aware documentation using AI models.

**Problem** 2: Developers struggle with code bugs that delay delivery.

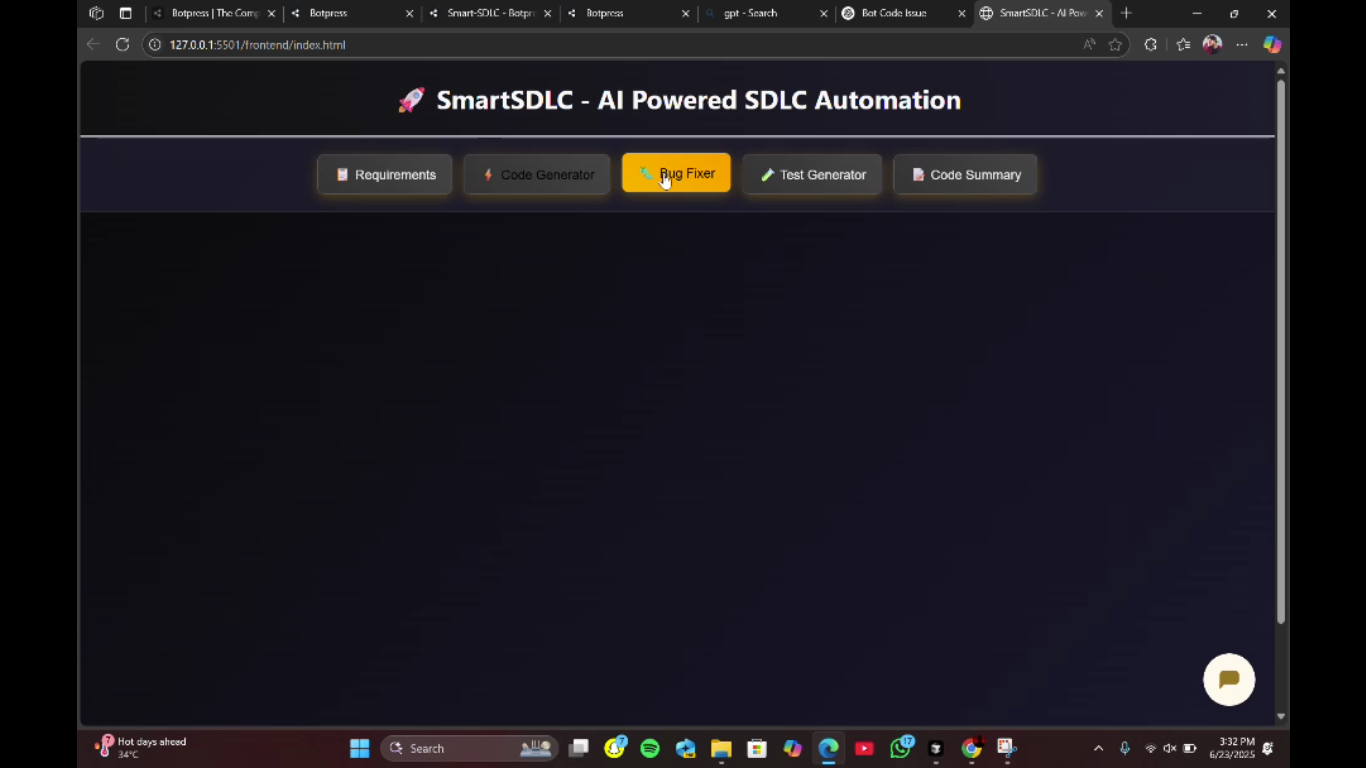
**Solution**: Integrate bug detection and fix suggestions using generative AI.

**Problem 3:** Writing new code from scratch increases development time.

**Solution:** Use AI to generate code snippets based on user intent.

**4.2 Proposed Solution**

Python developers in analyzing and debugging code efficiently, we have developed a Generative AI-powered SDLC Assistant tailored for Python environments. This AI model is designed to integrate into the Software Development Life Cycle and assist developers by providing intelligent, real-time support during code development and maintenance.



By automating code analysis and debugging, our solution reduces time spent on trial-and-error fixes, increases productivity, and supports cleaner, more maintainable codebases. Built using Python libraries such as Hugging Face Transformers, Pylint, and AST, and deployed via IBM Cloud, this assistant is scalable, secure, and easily integrable into existing Python development workflows.

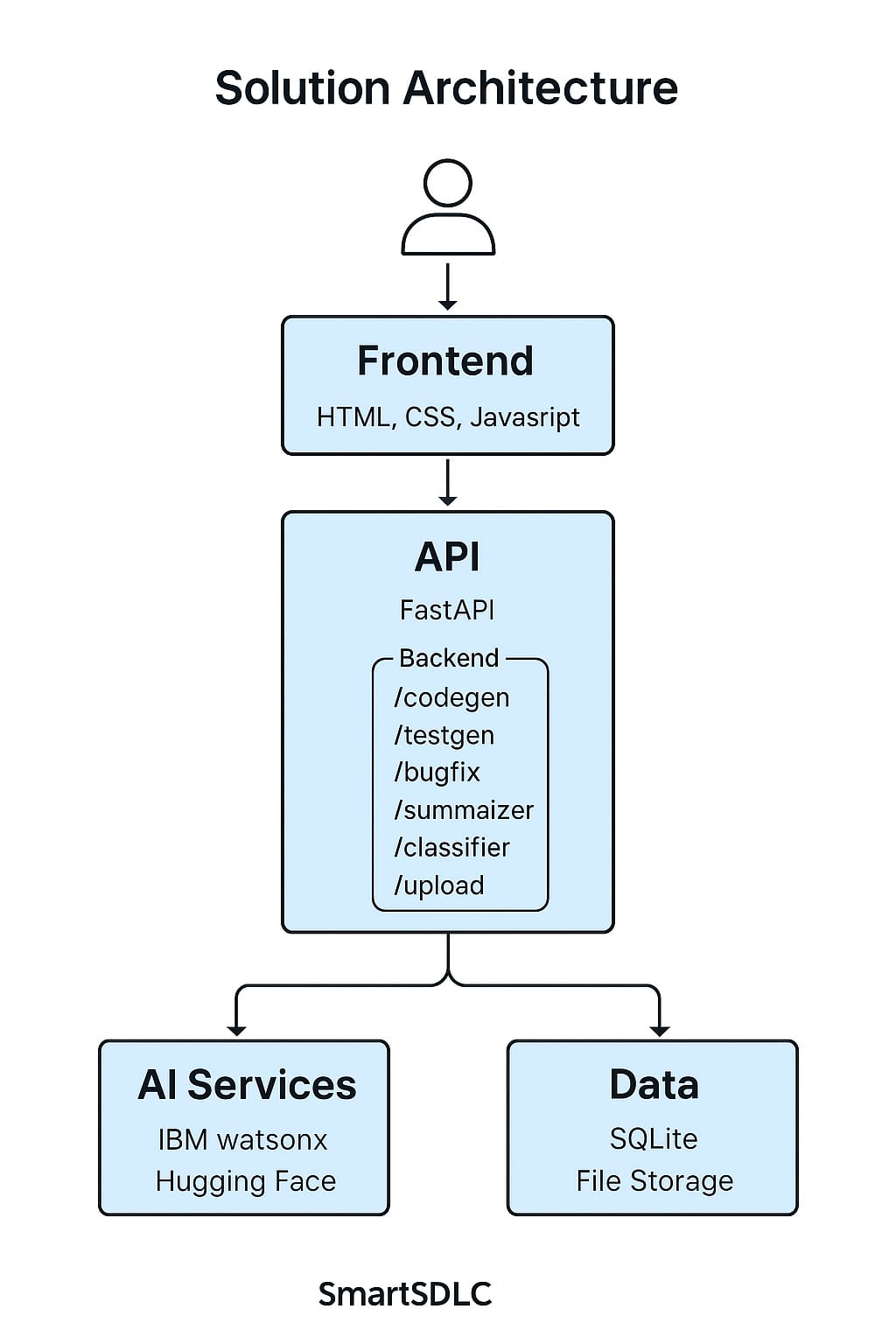
A screenshot of a computer

AI-generated content may be incorrect.

This Generative AI model ultimately serves as a smart pair programmer, allowing developers to focus on building robust features while relying on the assistant for faster debugging and code quality assurance.

**4.3 Solution Architecture:**

* **Developer**: Writes Python code in any IDE or interface
* **SDLC Assistant Interface**: UI/API layer to receive and send data
* **Analysis Module**: Parses and inspects code using Python tools

****

* **Generative AI**: Suggests fixes, explains errors using LLM
* **Deployment**: Runs on IBM Cloud for scalability and integration

APIs: IBM Cloud Watsonx API

\_API\_KEY=hf\_RbnLEyeUMGzyxzCXqYHoCfQWwTzrwhwDMl

\_MODEL\_ID=ibm/granite-20b-code-instruct

\_MODEL\_ID=bigcode/starcoder

**5.PROJECT PLANNING & SCHEDULING**

**5.1 Project Planning:**

Effective project planning is essential to ensure the timely and successful development of the Python-based Generative AI SDLC Assistant.

This section outlines the structured approach followed throughout the project lifecycle—from initial requirement gathering to final deployment and documentation.

Each phase is carefully designed with specific goals, deliverables, and timeframes to maintain clarity, accountability, and alignment with project objectives.

The planning ensures that development tasks are completed efficiently while maintaining high standards of quality, performance, and usability.

**Project Phases and Timeline:**

* Requirement Gathering (3 Days):  
  Define objectives, user needs, and project scope.
* Research & Feasibility (4 Days)  
  Analyze existing tools and generative AI techniques for Python code analysis.
* Model Design (5 Days)  
  Select model architecture (e.g., Transformer) and design the system pipeline.
* Dataset Preparation (5 Days)  
  Collect, clean, and annotate Python code samples for training.
* Model Development (7 Days)  
  Build and train the generative model; integrate Python analysis tools like AST and Pylint.
* Assistant Development (5 Days)  
  Develop the interface (CLI/API) and connect it to the AI backend.
* Testing & Evaluation (5 Days)  
  Test the assistant using real-world Python code; validate performance.
* Deployment (4 Days)  
  Deploy the solution on IBM Cloud with scalability and integration.
* Documentation (4 Days)  
  Prepare user guides, technical documentation, and architecture diagrams.
* Final Review (3 Days)  
  Conduct internal review, gather feedback, and present the final solution.

**6. FUNCTIONAL AND PERFORMANCE TESTING**

**Top of Form**

**⚙️ Functionality**

The Generative AI SDLC Assistant is designed to assist Python developers by automating key tasks in the Software Development Life Cycle, particularly during code development and debugging. The core functionalities include:

* Debugging Assistance: Detects common runtime issues and suggests possible fixes.
* Error Explanation: Translates error messages into understandable language for faster resolution.
* Code Suggestions: Offers optimized or corrected code snippets based on context.
* Real-Time Feedback: Integrates with the development workflow to provide immediate support during coding.

The assistant leverages Python libraries like ast, pylint, and transformer-based models to understand code structure and generate intelligent recommendations.

**Bottom of Form**

**✅ Testing**

To ensure reliability and performance, the assistant was tested through a structured process:

* **Unit Testing**: Individual components (code parser, error detector, AI model) were tested for expected functionality.
* **Functional Testing**: Validated that the assistant correctly identifies and responds to a variety of code issues.
* **Integration Testing**: Ensured smooth interaction between the AI engine, code analysis modules, and user interface.
* **User Scenario Testing**: Evaluated with multiple real-world Python scripts, covering edge cases, error patterns, and complex logic flows.
* **Performance Testing**: Measured response time and accuracy in providing suggestions under different code sizes.

Testing was carried out using sample Python projects and custom error cases to verify the assistant’s effectiveness in diverse scenarios.

**7. RESULTS**

**7.1 Output Screenshots:-**

**Code Generation:**

A screenshot of a computer

AI-generated content may be incorrect.

**Bug Fixing:**

A screenshot of a computer

AI-generated content may be incorrect.

**Document Analysis:**

A screenshot of a computer

AI-generated content may be incorrect.

**8.Advantages &Disadvantages**:

**✅ Advantages**

* Automates documentation, bug fixing, and code generation.
* Reduces developer workload and boosts productivity.
* Ensures uniform and readable technical documentation.
* Supports scalable AI processing through IBM Cloud.
* Offers a clean, simple HTML/CSS user interface.
* FastAPI provides efficient backend routing and response handling.
* Google Colab enables easy testing and demonstrations.
* Outputs available in multiple formats like Markdown, HTML, and PDF.
* Modular design makes it easy to add more features later.
* Useful for beginners and experienced developers alike.

**⚠️ Disadvantages**

* AI responses may require human review for accuracy.
* Performance depends on internet connection and cloud service latency.
* Increased usage may raise API and operational costs.
* Security needs careful attention when handling sensitive data.
* Initial setup across multiple platforms can be technically demanding.

**9. CONCLUSION:**

The Smart SDLc project is more than just a tech tool—it’s a reimagining of how software development gets done. By fusing the precision of IBM Cloud’s Instruct models with the elegance of FastAPI and the accessibility of a simple web interface, it empowers developers to hand off the tedious and reclaim their creativity.

From generating documentation that explains itself, to spotting bugs before they trip up a deadline, to translating natural language into clean code—Smart SDLc acts like a silent genius on the team, working tirelessly in the background.

Its modular design, real-time intelligence, and seamless Colab integration make it adaptable to solo coders and enterprise teams alike. Sure, it had challenges: wrangling APIs, managing latency, and tuning for relevance. But those weren't roadblocks they were milestones that sharpened its design.

In the end, Smart SDLc isn't just about efficiency—it's about elevating the craft of coding. A future where machines document, debug, and draft beside us isn’t far off. Thanks to Smart SDLc, it’s already to begun.

**10.FUTURE SCOPE:**

* **Support for Multiple Programming Languages**  
  Extend functionality beyond Python to include languages like Java, JavaScript, C++, and Go, making the assistant versatile for diverse development teams.
* **IDE & DevOps Integration**  
  Integrate the assistant into popular IDEs (e.g., VS Code, IntelliJ) and DevOps pipelines for real-time assistance during development, testing, and deployment stages.
* **Automated Test Case Generation**  
  Enable the assistant to generate unit tests and integration tests based on code analysis, improving test coverage and software reliability.
* **Security Vulnerability Detection**  
  Incorporate security scanning to detect potential vulnerabilities (e.g., insecure imports, input validation issues) early in the development process.
* **Natural Language Interface**  
  Add conversational capabilities so users can interact with the assistant using voice or text-based natural language queries.
* **Continuous Learning from User Feedback**  
  Use reinforcement learning or feedback loops to fine-tune the assistant’s suggestions based on real user interactions and corrections.
* **Team Collaboration Features**  
  Enable shared insights or suggestions in team environments, supporting collaborative debugging and pair programming.