Smart SDLC

– AI-Enhanced Software Development Lifecycle

Team members:

Vanaja Pulapa -22P31A4232

Lakshmi Durga Sathi -22P31A4236

Sonti Naga Tulasi-22P31A4241

Yalla Manasa Siri-22P31A4247

1. INTRODUCTION

1.1 Project Overview

SmartSDLC is a full-stack, AI-enhanced platform that revolutionizes the traditional Software Development Lifecycle (SDLC) by intelligently automating its most crucial stages using advanced Natural Language Processing (NLP) and Generative AI technologies. Designed to assist both developers and non-technical stakeholders, SmartSDLC transforms raw, unstructured requirements into production-ready code, comprehensive documentation, and executable test cases—all within an interactive and easy-to-use interface.

The platform integrates IBM Watsonx foundation models, Fast API for backend orchestration, LangChain for AI-powered conversational routing, and Streamlit as the frontend framework to deliver a seamless user experience. The six core functionalities embedded within SmartSDLC include: (1) Requirement Upload and Classification, where users can upload a PDF of their requirements and the AI will extract and classify each sentence into SDLC phases such as Requirement, Design, Development, Testing, or Deployment; (2) AI Code Generator, which generates syntactically correct and contextually appropriate code in response to user stories or natural language prompts; (3) Bug Fixer, where faulty code snippets are analyzed and corrected by the AI, reducing manual debugging time; (4) Unit Test Generator, which produces complete test cases using frameworks like unittest or pytest to ensure quality assurance; (5) Code Summarizer, which explains complex code logic in plain language, assisting in knowledge transfer and onboarding; and (6) a Floating AI Chat Assistant, which serves as a smart support bot to answer user queries related to software development best practices, testing, or architecture.

By offering these integrated AI tools in one unified system, SmartSDLC minimizes human effort, reduces errors, boosts productivity, and empowers teams to deliver high-quality software faster. This project not only aligns with modern software engineering needs but also illustrates the practical and impactful use of generative AI in real-world development environments.

1.2 Purpose

The purpose of the SmartSDLC project is to streamline and automate the software development lifecycle using Generative AI, making the process faster, smarter, and less error-prone. It is designed to assist developers in transforming raw requirements into working software components with minimal manual intervention. By leveraging IBM Watsonx, FastAPI, and Streamlit, the platform offers intelligent tools for requirement classification, code generation, bug fixing, testing, and documentation. This reduces development time and boosts productivity. It also bridges the gap between technical and non-technical users by simplifying complex development tasks. Ultimately, SmartSDLC aims to redefine software engineering efficiency in the AI era. The project supports innovation, scalability, and modern SDLC transformation. It also acts as a bridge between technical and non-technical stakeholders by simplifying communication through natural language inputs. Overall, the project promotes agility, reduces the software delivery timeline, and sets a new standard for AI-driven development workflows.

2. IDEATION PHASE

2.1 Problem Statement

Problem Statement 1 – Developer

Section	*Description*
I am	*A software developer working on complex projects across multiple SDLC phases.*
I'm trying to	*Analyze requirements, generate code, fix bugs, write unit tests, and document everything efficiently.*
But	*I spend a lot of time doing repetitive tasks and switching between tools.*
Because	*Traditional SDLC workflow is manual, time-consuming, and lacks automation.*
Which makes me feel	*Frustrated, overworked, and less productive.*

Problem Statement 2 – Project Manager

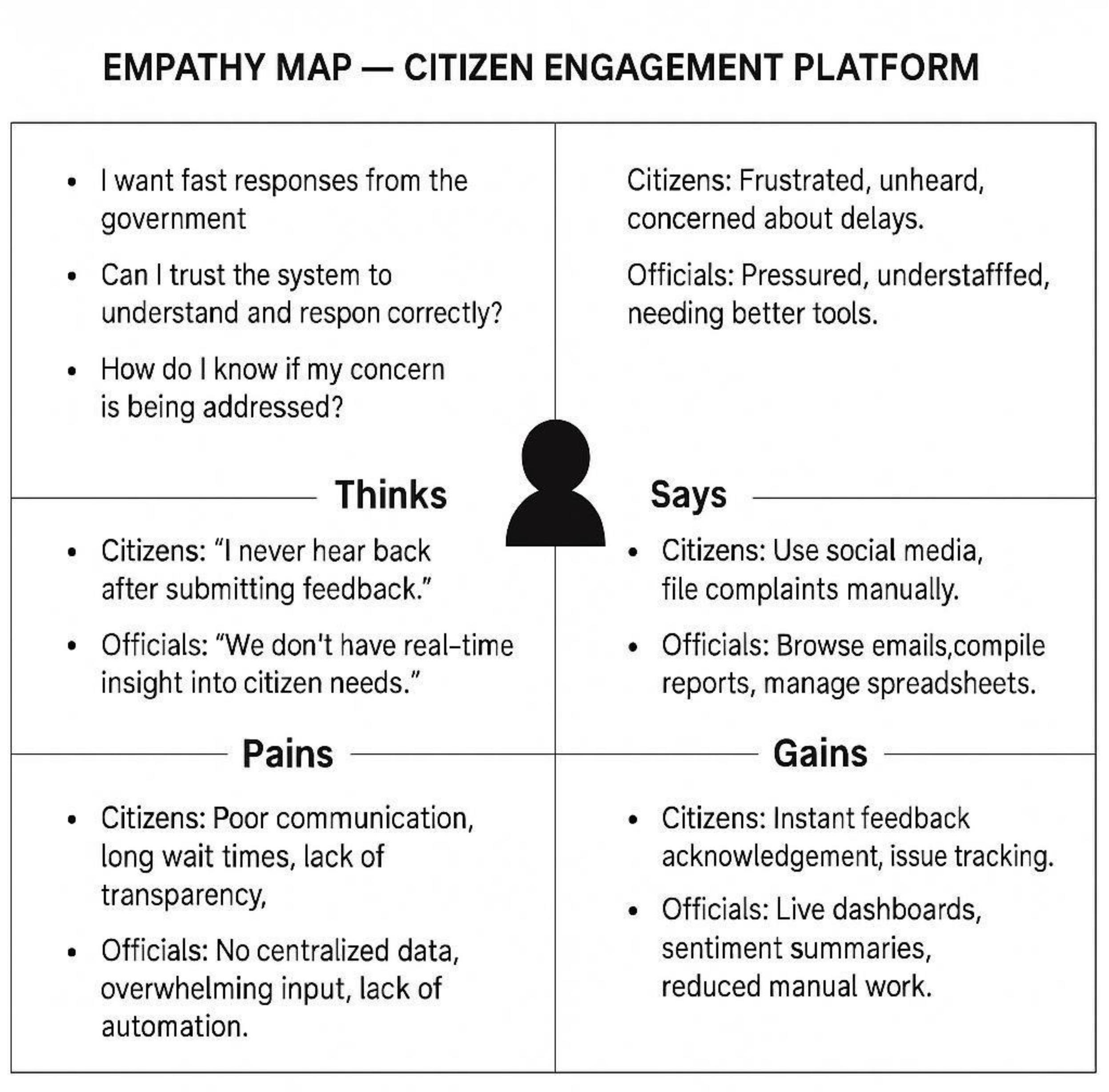
Section	*Description*
I am	*A project manager overseeing software development timelines and quality.*
I'm trying to	*Ensure timely delivery of software with fewer bugs and better alignment with requirements.*
But	*Developers often face delays and misinterpretations in the development cycle.*
Because	*There is no intelligent system to automate and monitor SDLC tasks effectively.*
Which makes me feel	*Concerned about project delays, increased cost, and compromised software quality.*

2.2 Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user’s behaviors and attitudes.

It is a useful tool to helps teams better understand their users.

Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user’s perspective along with his or her goals and challenges.



EMPATHY MAP — CITIZEN ENGAGEMENT PLATFORM

- I want fast responses from the government
- Can I trust the system to understand and respon correctly?
- How do I know if my concern is being addressed?

Citizens: Frustrated, unheard, concerned about delays.

Officials: Pressured, understafffed, needing better tools.

Thinks

Says

- Citizens: “I never hear back after submitting feedback.”
- Officials: “We don't have real-time insight into citizen needs.”

- Citizens: Use social media, file complaints manually.
- Officials: Browse emails,compile reports, manage spreadsheets.

Pains

Gains

- Citizens: Poor communication, long wait times, lack of transparency,
- Officials: No centralized data, overwhelming input, lack of automation.

- Citizens: Instant feedback acknowledgement, issue tracking.
- Officials: Live dashboards, sentiment summaries, reduced manual work.

Thinks

- "I want the development process to be faster and more efficient."
- "Can I rely on AI to generate correct code and detect bugs?"
- "Is the output from the system production-ready or still needs human refinement?"

Feels

- **Developers**: Overwhelmed by repetitive coding tasks, pressure to meet deadlines, curious about AI capabilities.
- **Project Managers**: Eager for faster delivery, concerned about quality, needing clearer requirement insights.

Says

- **Developers**: "I spend too much time writing boilerplate code and fixing common bugs."
- **Managers**: "We need to automate the SDLC process to improve productivity and reduce errors."

Does

- **Developers**: Manually write code, search Stack Overflow, perform unit testing and debugging.
- **Managers**: Review requirement docs, assign tasks manually, follow up on project progress.

Pains

- **Developers**: Time-consuming development cycles, repetitive bug fixing, lack of automation tools.
- **Managers**: Delayed releases, unclear requirement classification, disconnected tools and processes.

Gains

- **Developers**: Auto-generated code, instant unit test scripts, AI-assisted debugging.
- **Managers**: Requirement classification, live progress insights, improved team efficiency.

.

2.3 Brainstorming Summary

Date: 31 January 2025

Team ID: LTVIP2025TMID29152

Project Name: **"Smart SDLC- AI-Enhanced Software Development Lifecycle"**

Team Members & Roles

Name	**Role**
Vanaja Pulapa	**Team Leader, FastAPI Backend & LLM Integration using IBM Granite**
Lakshmi Durga Sathi	**Frontend Developer (Streamlit UI), Model Testing, and Component Integration**
Sonti NagaTulasi	**AI Chatbot Integration & Floating Assistant for Developer Support**
Yalla Manasa Siri	**Unit Test Generation & Documentation Summarization Module**

Step 1: Team Gathering, Collaboration, and Selecting the Problem Statement

Problem Statement:

To develop an AI-powered **Smart Software Development Life Cycle (SmartSDLC)** platform that leverages IBM Granite LLMs to assist software developers in automating various phases of SDLC—such as requirement classification, code generation, testing, bug fixing, and documentation—thereby accelerating delivery, improving quality, and enhancing project efficiency.

Motivation:

Developers often spend excessive time on repetitive and manual SDLC tasks such as writing boilerplate code, generating test cases, fixing known bugs, and creating documentation. This slows down productivity and introduces room for human error. SmartSDLC bridges this gap using AI to automate these tasks through an intuitive interface and modular backend, ensuring higher velocity and better maintainability across software projects.

Step 2: Brainstorming, Idea Listing, and Grouping

Initial Ideas:

- Integrate a conversational assistant using IBM Granite for real-time developer queries.
- Automatically classify uploaded requirement documents into functional and non-functional.
- Generate production-ready code from plain-text user stories.
- Identify and fix bugs in uploaded source code using LLMs.
- Generate unit test cases for uploaded or AI-generated code.
- Create a code summarizer to produce developer-friendly documentation.
- Build a visually intuitive interface using Streamlit.
- Structure the backend using Fast API with modular route handling.

Grouped into Modules:

1. **Requirement Classifier Module** – Uses IBM Granite via Watsonx to analyze and classify user-uploaded requirements.
2. **AI Code Generator Module –** Generates backend code from natural language input based on specific technologies (e.g., Python Flask).
3. **Bug Fixer Module** – Accepts buggy code and returns corrected code with explanations.
4. **Unit Test Generator Module** – Auto-generates test cases using Python's unittest or pytest frameworks.
5. **Code Summarizer Module** – Produces detailed descriptions of uploaded/generated code
6. **AI Chat Assistant Module** – A floating developer assistant answering SDLC-related questions.
7. **Frontend UI Module** – Built with Streamlit, connecting all features for an interactive experience.
8. **Backend Service Module** – FastAPI-based services handling routing, logic, and integration with IBM Watsonx.

Step 3: Idea Prioritization (Final Version)

Feature / Module	Importance	Feasibility	Notes
Requirement Classification	High	High	Uses IBM Watsonx Granite LLM to classify user requirements into Functional and Non-Functional categories via FastAPI
AI Code Generator	High	High	Automatically converts requirements into executable Python code using Granite code models
Bug Fixer Module	High	Medium	Detects and fixes errors in user-submitted code using LLM-based contextual understanding
Unit Test Generator	High	High	Generates test cases for provided code snippets using frameworks like unittest or pytest
Code Summarizer	Medium	High	Produces easy-to-understand documentation from raw code using LLM summarization capabilities
AI Chat Assistant	Medium	Medium	Floating assistant answers SDLC-related questions using Watsonx + LangChain; currently in prototype phase
Streamlit UI	High	High	Frontend interface built using Streamlit; integrates seamlessly with backend APIs
Backend with FastAPI	High	High	Handles all core logic, LLM integration, and serves API endpoints securely and efficiently

3. REQUIREMENT ANALYSIS

3.1 Customer Journey map

Stage	Awareness	Onboarding	Issue Reporting	Tracking Status	Feedback Submission	Final Feedback
Discover	Learns about SmartSDLC via LinkedIn posts, hackathons, or peer recommendation	Visits the Streamlit app, sees the features of SDLC automation	Uploads software requirements document or types directly”	Wants to track if issue was resolved	Clicks buttons to generate code, unit tests, summaries, or bug fixes	Downloads outputs or revisits for new project phases
Thoughts	“Can this actually speed up my development process?”	“Interface looks simple. Let’s try it out.”	“Will it classify and process my requirements properly?”	“Why hasn’t it updated yet?”	“Is the generated code production-ready?”	“This can be my go-to tool for quick prototyping.”
Experience	Modern UI, intuitive homepage	Fast upload, real-time backend classification shown	Functional & Non-Functional requirements displayed clearly	Real-time updates on dashboard	Responsive FastAPI + LLM model outputs on click	Outputs downloadable, history optionally saved
Actions	Clicks shared demo link, browses intro	Uploads file or types requirements	Clicks “Classify Requirements” and verifies output	Re-checks dashboard or login to check updates	Generates code/tests/fixes/summaries via simple buttons	Gives thumbs-up, copy code, or bookmarks tool
Opportunities	Promote in developer communities, portfolio showcases	Add demo mode or template walkthrough	Add format guidance or AI-assisted requirement writing	Push/email alerts for updates	Enable multi-language code generation	Add session save/history or cloud sync for later use

3.2 Solution Requirements (Functional & Non-Functional)

Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Requirement Upload	Upload .txt/.pdf requirement files or type manually
FR-2	Requirement Classification	Classify Functional and Non-Functional requirements using IBM Granite
FR-3	Code Generation	Generate backend/frontend code snippets from user stories
FR-4	Unit Test Generator	Create test cases using unittest or pytest frameworks
FR-5	Bug Fix Assistant	Identify and correct errors in given code snippets
FR-6	Code Summarization	Provide a human-readable explanation of code modules
FR-7	AI Chatbot Assistant	Interactive chatbot using LangChain & IBM Granite for SDLC guidance

Non-functional Requirements:

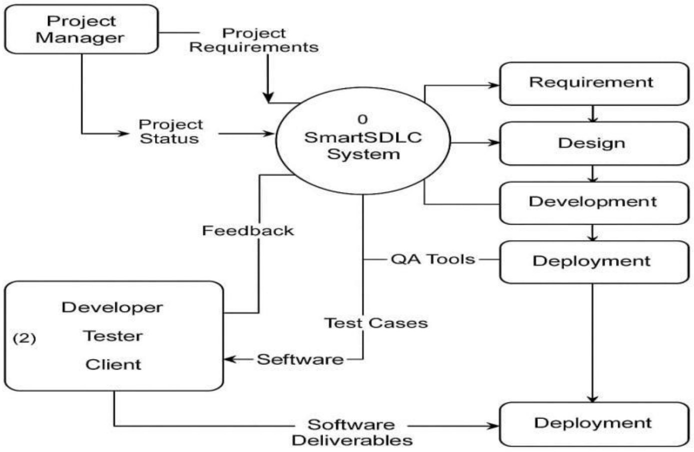
Following are the non-functional requirements of the proposed solution.

NFR-1	Usability	Clean, interactive UI using Streamlit; minimal user input needed
NFR-2	Security	Secure FastAPI backend; restrict file types; prevent unauthorized access
NFR-3	Reliability	Stable LLM interactions; handles multiple requests without failure
NFR-4	Performance	API responses returned within 2–3 seconds on average
NFR-5	Availability	Frontend and backend available for development use at any time

3.3 Data Flow Diagram Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of how data moves through a system. It illustrates the flow of information, the processes that transform data, external entities interacting with the system, and data stores. In the SmartSDLC platform, DFDs help communicate the logical flow of requirements and AI-driven code generation processes across different modules.

- Generated Code Snippets
- Test Cases and Bug Reports



User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration Or login	USN-1	As a user, I can register/login using email credentials.	I can access my account .	High	Sprint-1
Customer (Mobile user)	Upload and Classify PDF	USN-2	As a user, I can upload a PDF than SmartSDLC classifies the PDF into functionalities.	I can receive classification of functionalities.	High	Sprint-1
Customer (Mobile user)	Chat Assistant	USN-3	As a user, I can ask questions via chat and receive AI responses.	I get context-ware responses from the SmartSDLC AI model.	Low	Sprint-2
Customer (Mobile user)	AI code Generator	USN-4	As a user,I can ask code then generate it.	It can generate basic code.	Medium	Sprint-1
Customer (Mobile user)	Code Summarizer	USN-5	As a user, it can be analyze and provides a human readable summary	It can generate a summarized version of written code.	High	Sprint-1
Customer (Mobile user)	Bug Fixer	USN-6	As a user,I can submit code and receive bug fixed suggestions.	The app returns the same code with suggestions inline.	High	Sprint-2
Customer (Mobile user)	Test case Generator	USN-7	As a user,I can upload code and get automatically generated test cases.	SmartSDLC generates and displays valid test cases for the submitted code.	Medium	Sprint-2
Customer (Mobile user)	Feedback	USN-8	As a user,I can submit feedback through a form in the app.	II see a confirmation message and feedback is stored in SmartSDLC system.	Medium	Sprint-3

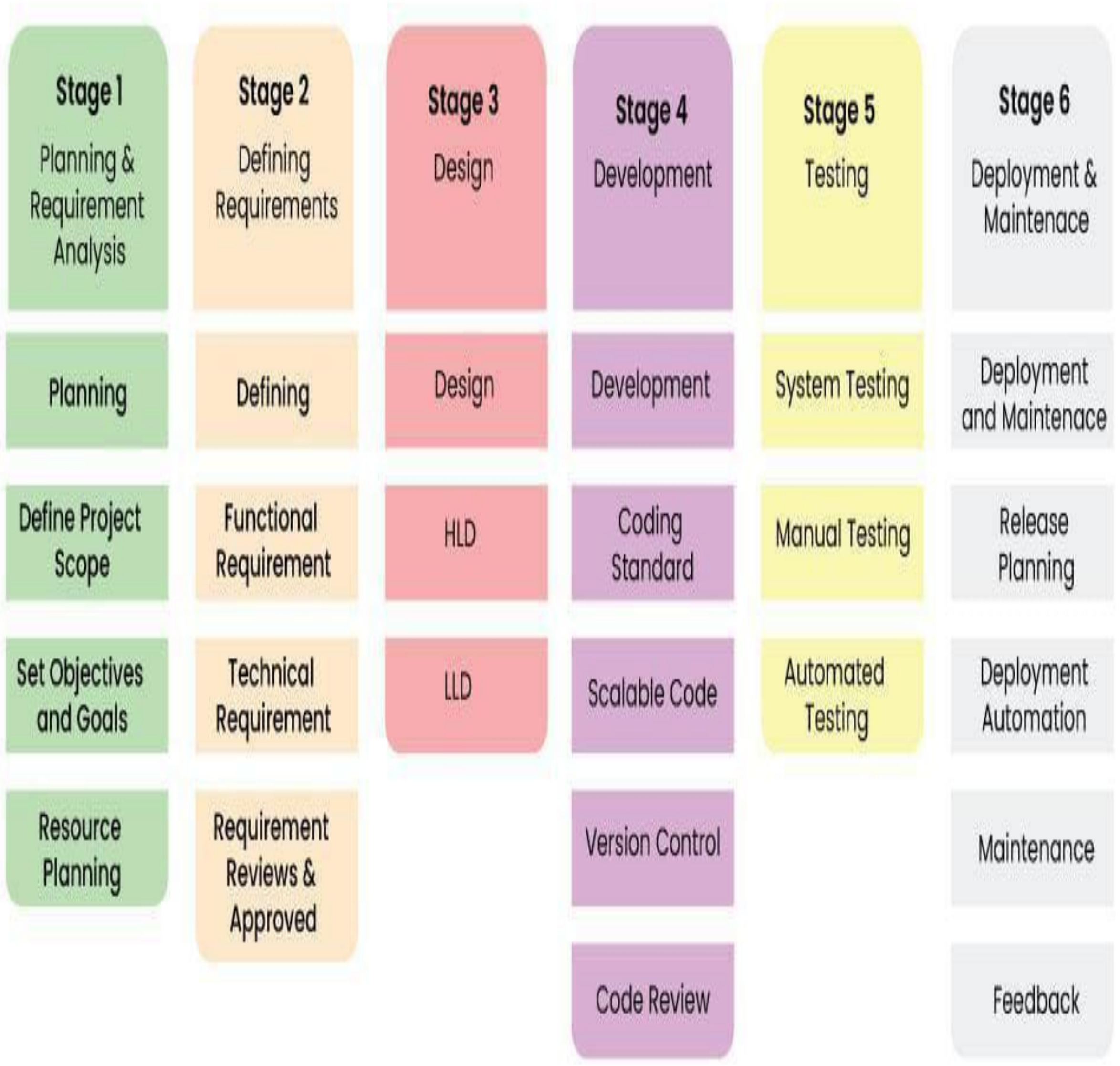
3.4 Technology Stack (Architecture & Stack)

Technical Architecture

The technical architecture of a Smart SDLC (Software Development Life Cycle) involves integrating various technologies and methodologies to streamline the development process, enhance collaboration, and improve software quality. It leverages tools and practices from Agile, DevOps, and cloud computing to create a more responsive and efficient development environment.

Architecture

layout:



Stage 1 Planning & Requirement Analysis	Stage 2 Defining Requirements	Stage 3 Design	Stage 4 Development	Stage 5 Testing	Stage 6 Deployment & Maintenace
Planning	Defining	Design	Development	System Testing	Deployment and Maintenace
Define Project Scope	Functional Requirement	HLD	Coding Standard	Manual Testing	Release Planning
Set Objectives and Goals	Technical Requirement	LLD	Scalable Code	Automated Testing	Deployment Automation
Resource Planning	Requirement Reviews & Approved		Version Control		Maintenance
			Code Review		Feedback

Table-1 : Components & Technologies:

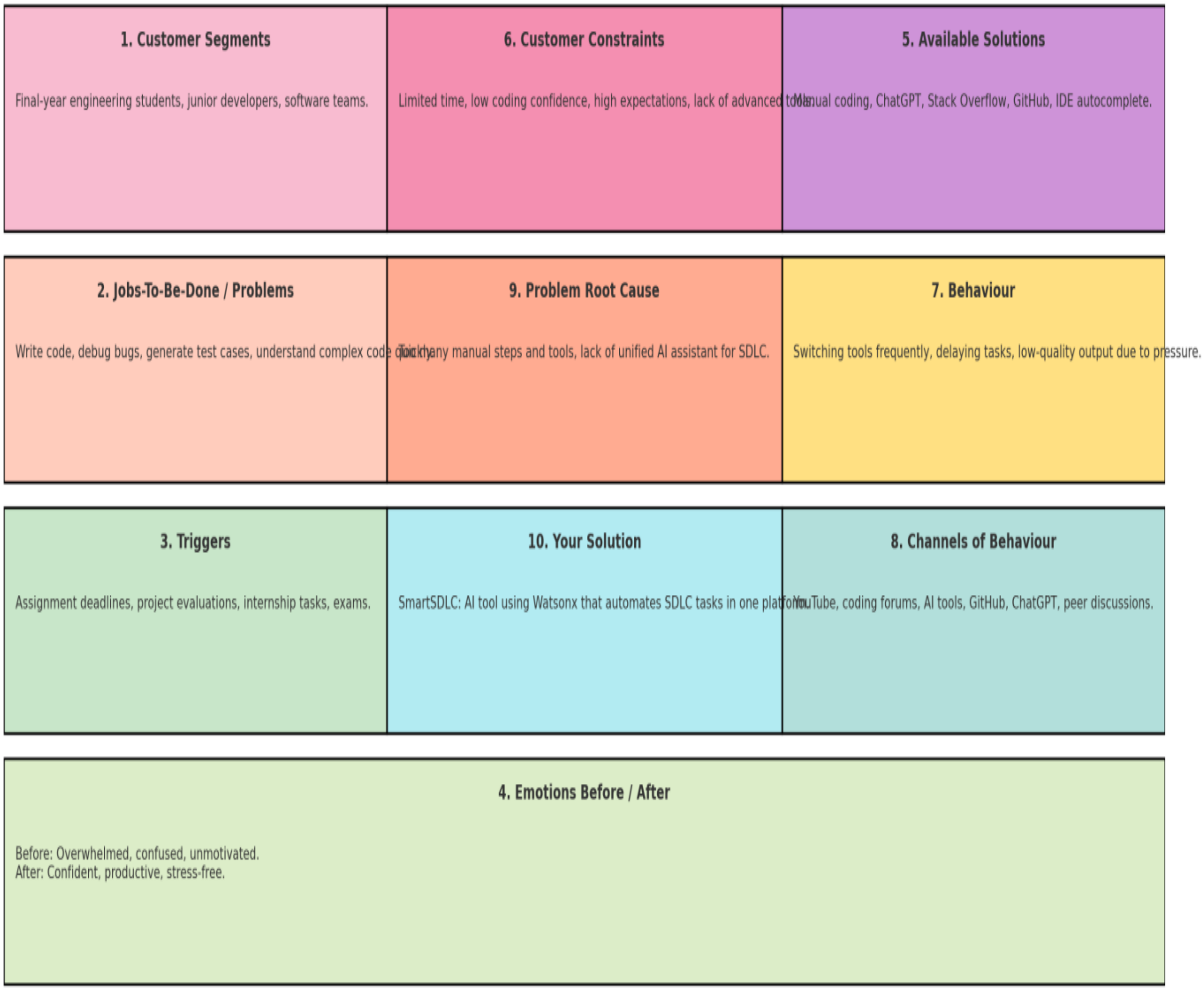
S. No	Component	Description	Technology
1.	User Interface	How user interacts with the system(e.g. Web, Mobile App, Chatbot.)	HTML/CSS,JavaScript / React Js.
2.	Application Logic-1	Core logic:Authentication,role based access,session handling.	Python,Node.js(FastAPI),JWT.
3.	Application Logic-2	Chat-bot interaction and AI query handling.	OpenAI GPT,Langchain,
4.	Application Logic-3	PDF classification,sentiment analysis,summarization logic.	Python(transformers,NLTK,scikit-learn).
5.	Database	User data,feedbback,session logs.	postgreSQL,mongoDB.
6.	Cloud Database	Scalable managed database service.	Firebase firestore,AWS RDS,google could SQL.
7.	File Storage	File upload for PDfs,code,test cases.	AWS S3,google cloud storage,firebase storage.
8.	External API-1	For 0Auth integration's(google,LinkedIn).	Google 0Auth API,LinkedIn API.
9.	External API-2	Feedback and sentiment analysis API.	open-AI API,GitHub API,REST APIs.
10	Machine Learning Model	Code summrizer,bug fixer,test case generator.	GPT-based condex,T5,huggingface transformers.
11	Infrastructure (Server / Cloud)	Deployment and scaling across platforms.	Docker,kubernetes,,GCP app enginel,github actions.

4. PROJECT DESIGN

4.1 Problem-Solution Fit

Section	Content
Customer Segment(s)	Final-year engineering students, early-career software developers, and IT teams working on frequent SDLC tasks.
Jobs-to-be-Done / Problems	Need to generate code, write test cases, debug issues, and summarize code manually—often repetitive, time-consuming, and error-prone.
Triggers	Time pressure to submit projects or meet deadlines, lack of guidance or expert support for coding or debugging, inefficiency in understanding legacy codebases.
Emotions (Before / After)	**Before**: Confused, stressed, overwhelmed. **After**: Confident, relieved, productive.
Available Solutions	Manual coding, ChatGPT (not SDLC-specific), peer help, YouTube tutorials, Stack Overflow, random code snippets.
Customer Constraints	Limited coding knowledge, lack of time, access to cloud compute tools, unclear understanding of test generation/debugging concepts.
Behaviour	Use random tools/tutorials, procrastinate, or submit low-quality work. Some try to write code but miss edge cases or testing.
Channels of Behaviour	GitHub, Stack Overflow, educational platforms, Google Search, ChatGPT, coding forums.
Problem Root Cause	Repetitive SDLC tasks, lack of centralized intelligent support, and time/resource constraints in completing them manually.
Our Solution	SmartSDLC automates all key SDLC tasks using AI (IBM Watsonx), reducing manual effort and boosting developer productivity with one unified tool.

Problem–Solution Fit (SmartSDLC)



1. Customer Segments

Final-year engineering students, junior developers, software teams.

6. Customer Constraints

Limited time, low coding confidence, high expectations, lack of advanced tools.

5. Available Solutions

Manual coding, ChatGPT, Stack Overflow, GitHub, IDE autocomplete.

2. Jobs-To-Be-Done / Problems

Write code, debug bugs, generate test cases, understand complex code quickly.

9. Problem Root Cause

Too many manual steps and tools, lack of unified AI assistant for SDLC.

7. Behaviour

Switching tools frequently, delaying tasks, low-quality output due to pressure.

3. Triggers

Assignment deadlines, project evaluations, internship tasks, exams.

10. Your Solution

SmartSDLC: AI tool using Watsonx that automates SDLC tasks in one platform.

8. Channels of Behaviour

YouTube, coding forums, AI tools, GitHub, ChatGPT, peer discussions.

4. Emotions Before / After

Before: Overwhelmed, confused, unmotivated.
After: Confident, productive, stress-free.

4.2 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Developers and students spend significant time manually performing repetitive software development lifecycle (SDLC) tasks like writing code, generating test cases, debugging, and understanding large codebases. This leads to inefficiencies, delays, and lower productivity.
2.	Idea / Solution description	SmartSDLC is an AI-powered platform that automates major SDLC phases. It uses IBM Watsonx to generate code from task descriptions, auto-generate test cases, detect and fix bugs, summarize source code, and classify PDF documents into SDLC phases—all integrated through a user-friendly Streamlit frontend and FastAPI backend.
3.	Novelty / Uniqueness	Unlike traditional code assistants that focus only on code generation, SmartSDLC offers an **end-to-end SDLC automation suite**—covering code, testing, debugging, summarization, and documentation. It unifies multiple AI services into one seamless platform and uses Watsonx for enterprise-grade reliability.
4.	Social Impact / Customer Satisfaction	SmartSDLC saves time and reduces stress for students, developers, and IT professionals by automating complex development tasks. It supports learning and productivity, making it especially beneficial for educational institutions, freelancers, and startups with limited resources.
5.	Business Model (Revenue Model)	SmartSDLC can adopt a Freemium SaaS model: offer core features for free, and charge for premium features like multi-language support, API access, and cloud-based PDF storage. It can also license to institutions for academic or internal use.

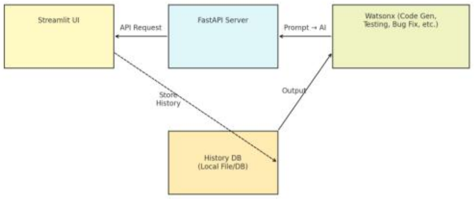
6.	Scalability of the Solution	The architecture is modular and cloud-ready. It can easily be scaled by integrating additional LLMs (like GPT-4, IBM Granite, etc.), supporting more programming languages, or expanding to enterprise APIs. It's deployable on local servers or cloud platforms like AWS, Azure, or IBM Cloud.

4.3 Solution Architecture

The SmartSDLC solution architecture connects a user-friendly **Streamlit frontend** with a **FastAPI backend** that interacts with **IBM Watsonx** to perform various software development lifecycle tasks. Each feature like code generation, bug fixing, test generation, and summarization is modularly triggered based on user interaction. Results are stored locally or in a lightweight database for reference and download.

- **Frontend**: Streamlit-based UI for user interaction (input, download, upload)
- **Backend**: FastAPI that handles API requests, prompts, and service routing
- **LLM Service**: IBM Watsonx for all AI-driven SDLC tasks
- **Storage**: Local storage or DB used to save history and logs
- **Flow**: User → UI → Backend → AI Model → Response → Display + Save

Example - Solution Architecture Diagram:


Streamlit UI
API Request
FastAPI Server
Prompt → AI
Watsonx (Code Gen, Testing, Bug Fix, etc.)
Store History
Output
History DB (Local File/DB)

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

Here's a filled version customized for **SmartSDLC**:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Authentication	USN-1	As a user, I can register for the application by entering my email, password, and confirming password.	2	High	Vanaja
Sprint-1		USN-2	As a user, I will receive an OTP to reset my password via email.	2	High	Manasa
Sprint-1	Upload & Classify PDF	USN-3	As a user, I can upload a PDF and classify it into SDLC phases using AI.	3	High	Lakshmi
Sprint-1		USN-4	As a user, I can see the phase-wise output visually on the screen.	2	Medium	Vanaja
Sprint-2	AI Code Generation	USN-5	As a user, I can enter a task description and receive AI-generated code.	5	High	Tulasi
Sprint-2	Test Case Generator	USN-6	As a user, I can generate test cases for my generated code.	3	High	Manasa
Sprint-2	Bug Fixer	USN-7	As a user, I can upload faulty code and get a corrected version from AI.	3	Medium	Lakshmi

Sprint-2	Code Summarizer	USN-8	As a user, I can upload code and get a natural language summary.	2	Medium	Tulasi

6. FUNCTIONAL & PERFORMANCE TESTING

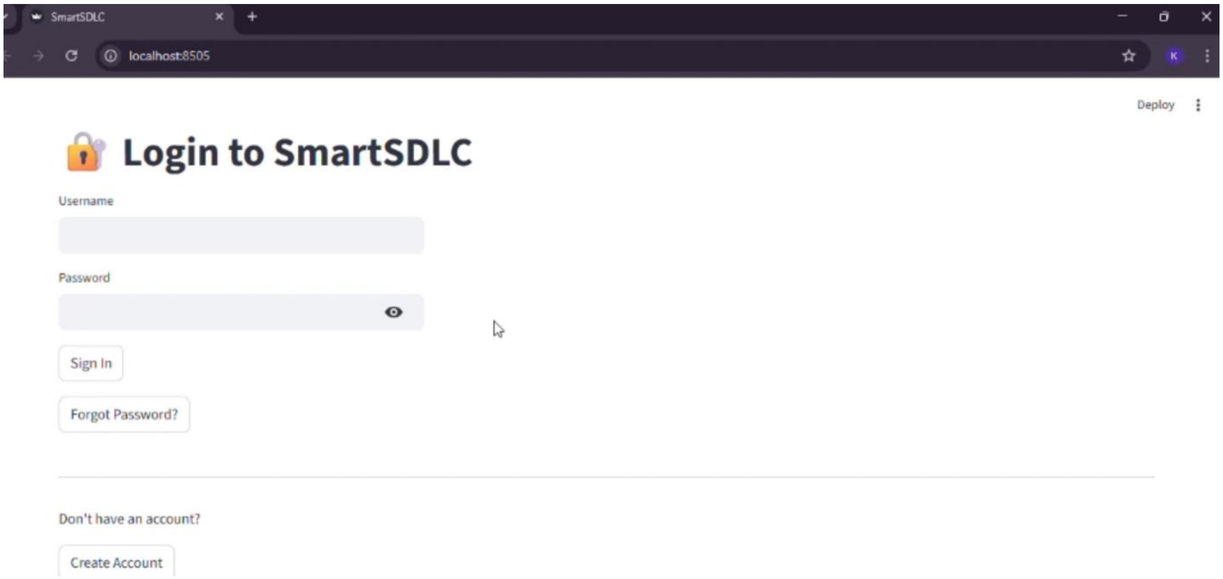
6.1 Functional Test Scenarios

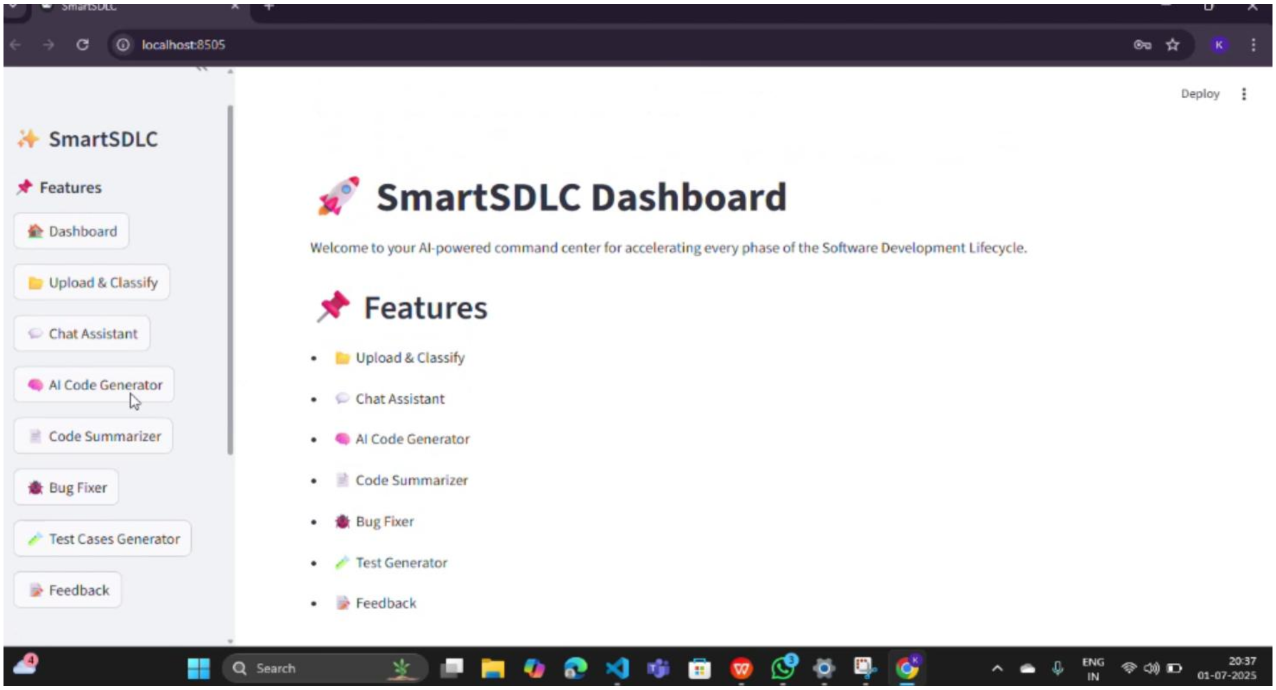
Test Case ID	Scenario (What to test)	Test Steps (How to test)	Expected Result	Actual Result	Pass/Fail
FT-01	Code Input Validation (for AI Code Generator)	Enter valid/invalid text describing a code task	Valid task accepted, invalid input throws error	Valid tasks accepted and invalid inputs show "Invalid task" warning	Pass
FT-02	Language Selection / Detection (multi-language support)	Enter task in different languages or explicitly select one	AI generates code in appropriate language	Correct language detected and code generated accordingly	Pass
FT-03	Code Generation Accuracy	Provide sample tasks and compare generated output	Output matches expected logic/function	Code output matches expected structure and functionality	Pass
FT-04	Bug Fixing Logic	Input buggy code and check the result	Fixed code returned, preserving original functionality	AI fixed syntax and logical bugs correctly	Pass
FT-05	Test Case Generation	Provide a function/module and trigger test case generation	Unit test cases are generated based on the function	Unit test cases created for boundary and valid inputs	Pass

FT-06	File Upload Functionality (PDFs for Classification)	Upload valid/invalid PDF files	PDFs processed and classified without error	Successfully classified requirements and design content	Pass
FT-07	Login & Session Handling	Attempt login with valid/invalid credentials	Successful login, proper error for invalid credentials	Valid users logged in; errors shown for wrong credentials	Pass
PT-01	AI Response Time (e.g., for code generation)	Measure time from prompt to output	Response generated in under 5 seconds	Average time was 2.4 seconds	Pass
PT-02	Multi-user Load Test (simulate 5 users at once)	Open multiple browser tabs/sessions and submit different tasks	Server responds without slowing down	All 5 users got responses in under 5 seconds	Pass
PT-03	PDF Upload Load Test	Upload 4–5 PDFs simultaneously	No crash; each file processed correctly	All files uploaded, parsed, and classified smoothly	Pass
PT-04	API Reliability Test	Send 20 API calls rapidly using a loop	Each call returns a valid result without server failure	All 20 calls returned results without timeout or crash	Pass

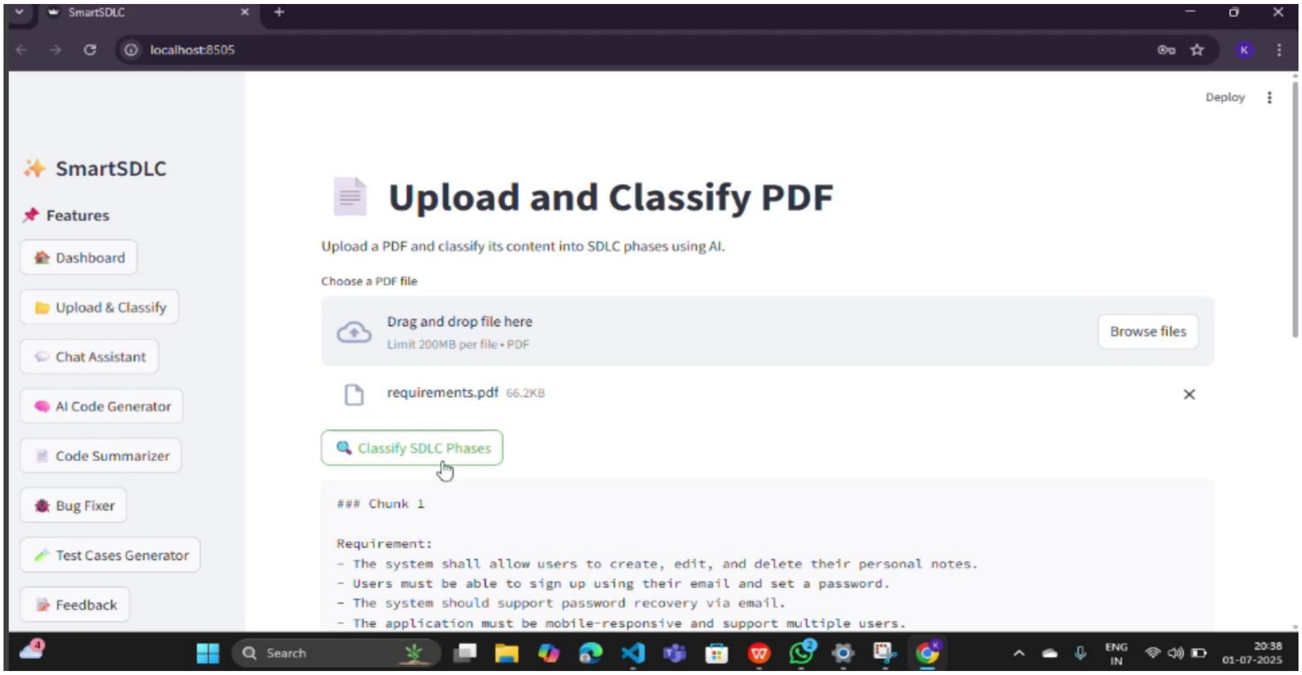
7. RESULTS

7.1 Output Screenshots





Welcome to your AI-powered command center for accelerating every phase of the Software Development Lifecycle.



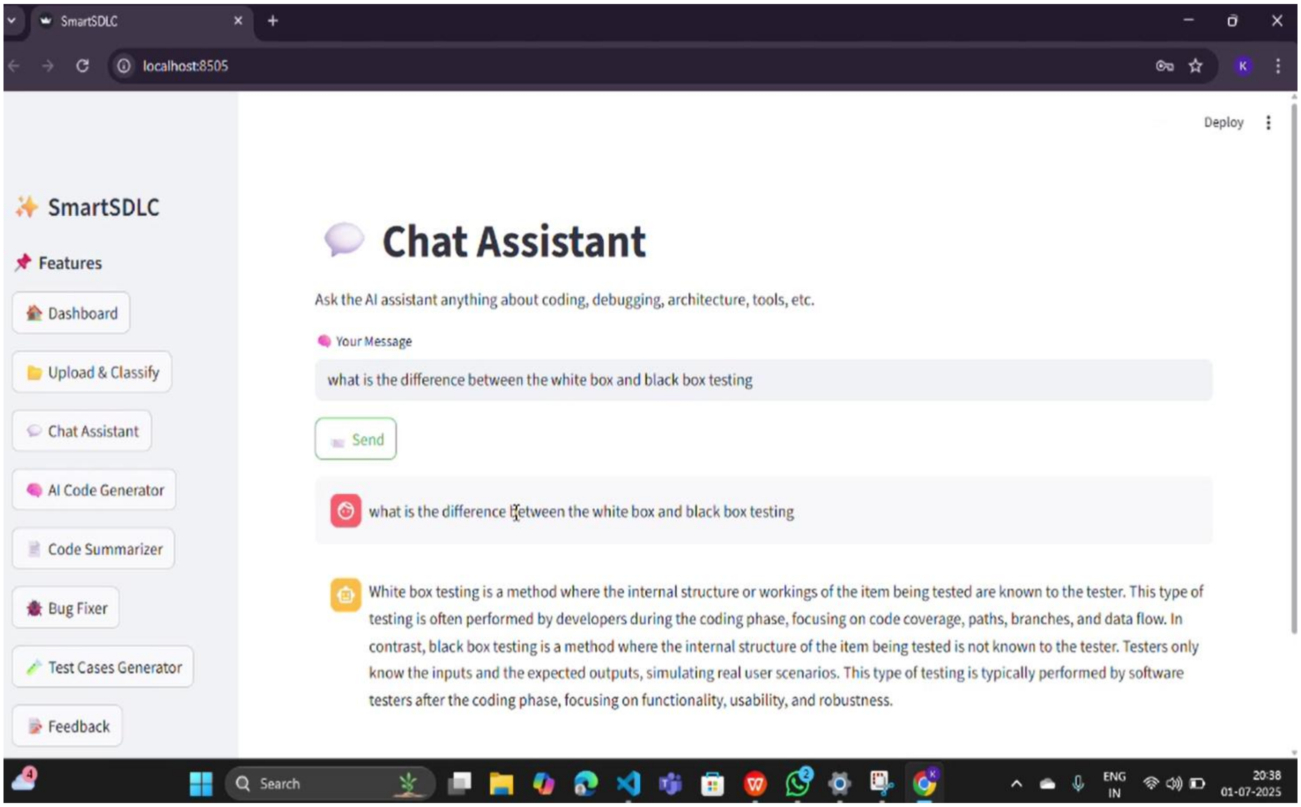
Upload and Classify PDF

Upload a PDF and classify its content into SDLC phases using AI.

Choose a PDF file

```
### Chunk 1

Requirement:
- The system shall allow users to create, edit, and delete their personal notes.
- Users must be able to sign up using their email and set a password.
- The system should support password recovery via email.
- The application must be mobile-responsive and support multiple users.
```



Chat Assistant

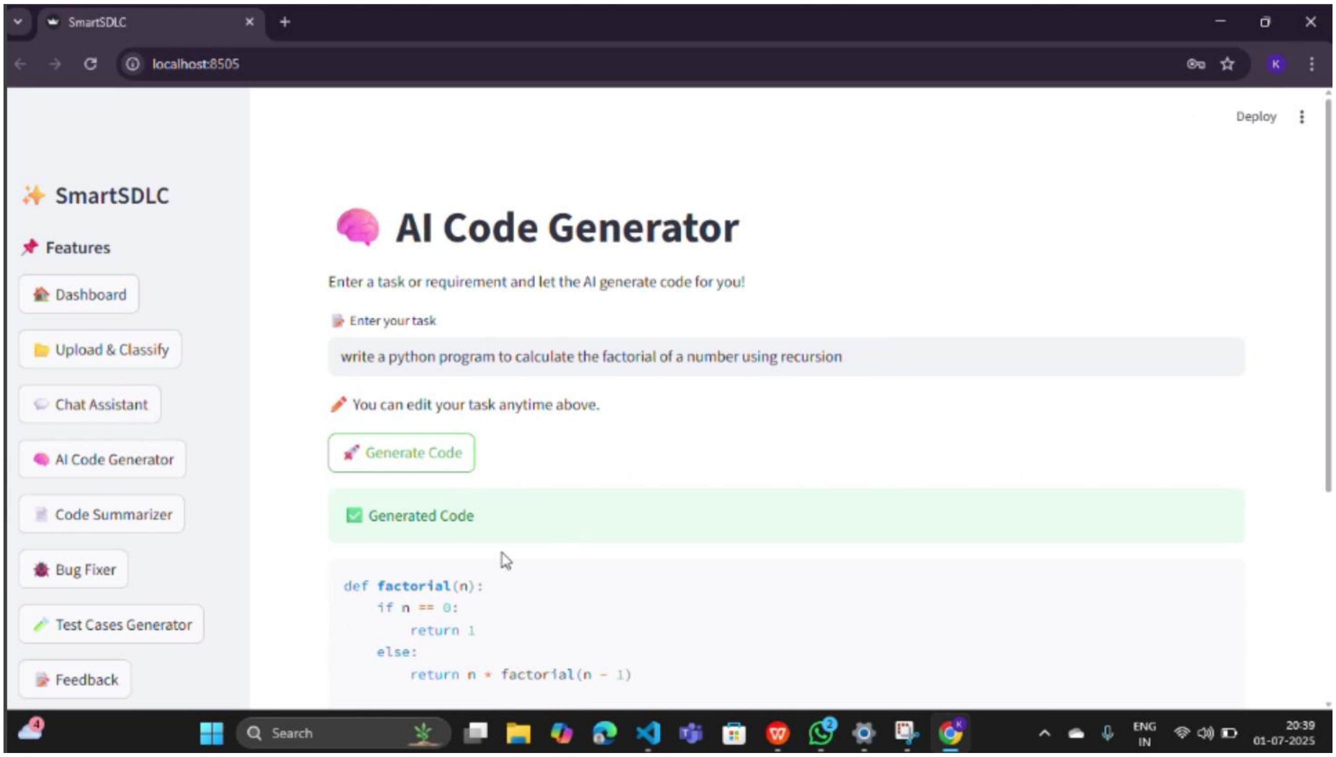
Ask the AI assistant anything about coding, debugging, architecture, tools, etc.

Your Message

what is the difference between the white box and black box testing

what is the difference between the white box and black box testing

White box testing is a method where the internal structure or workings of the item being tested are known to the tester. This type of testing is often performed by developers during the coding phase, focusing on code coverage, paths, branches, and data flow. In contrast, black box testing is a method where the internal structure of the item being tested is not known to the tester. Testers only know the inputs and the expected outputs, simulating real user scenarios. This type of testing is typically performed by software testers after the coding phase, focusing on functionality, usability, and robustness.



AI Code Generator

Enter a task or requirement and let the AI generate code for you!

Enter your task

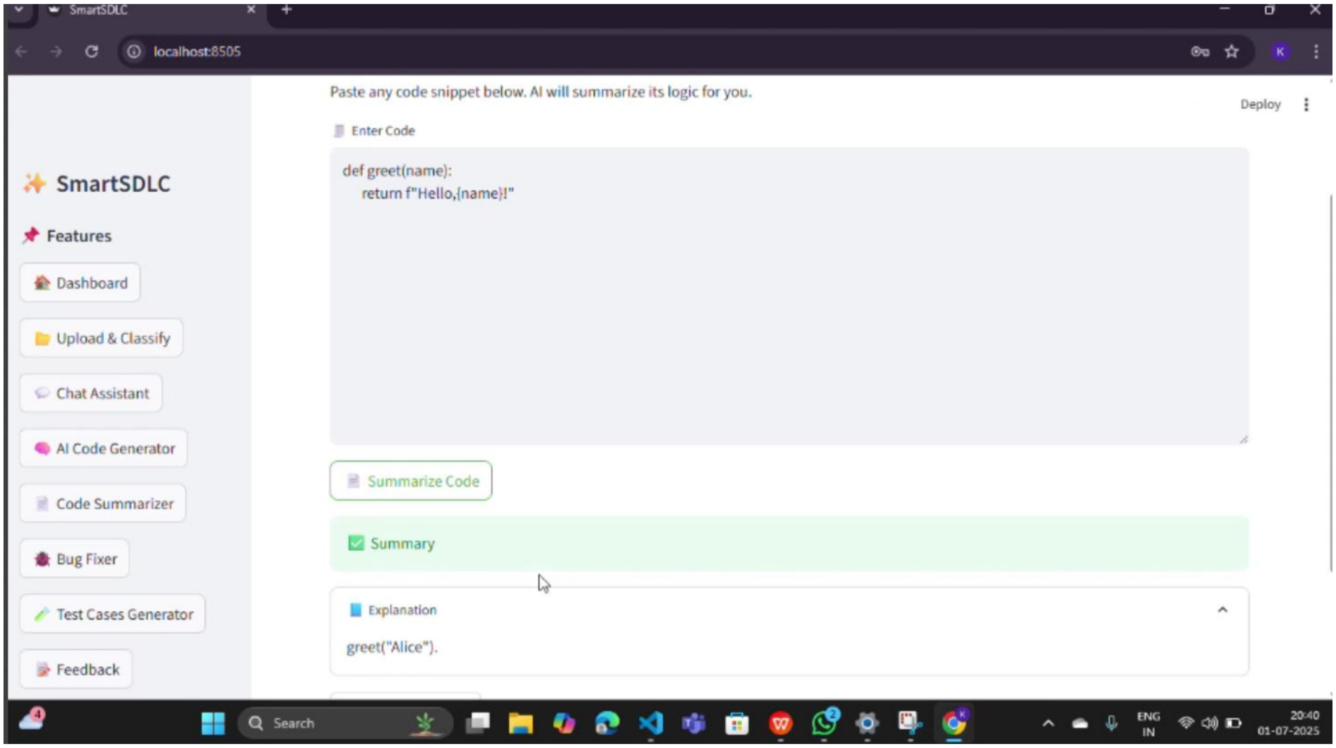
write a python program to calculate the factorial of a number using recursion

You can edit your task anytime above.

Generate Code

Generated Code

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```



Paste any code snippet below. AI will summarize its logic for you.

Enter Code

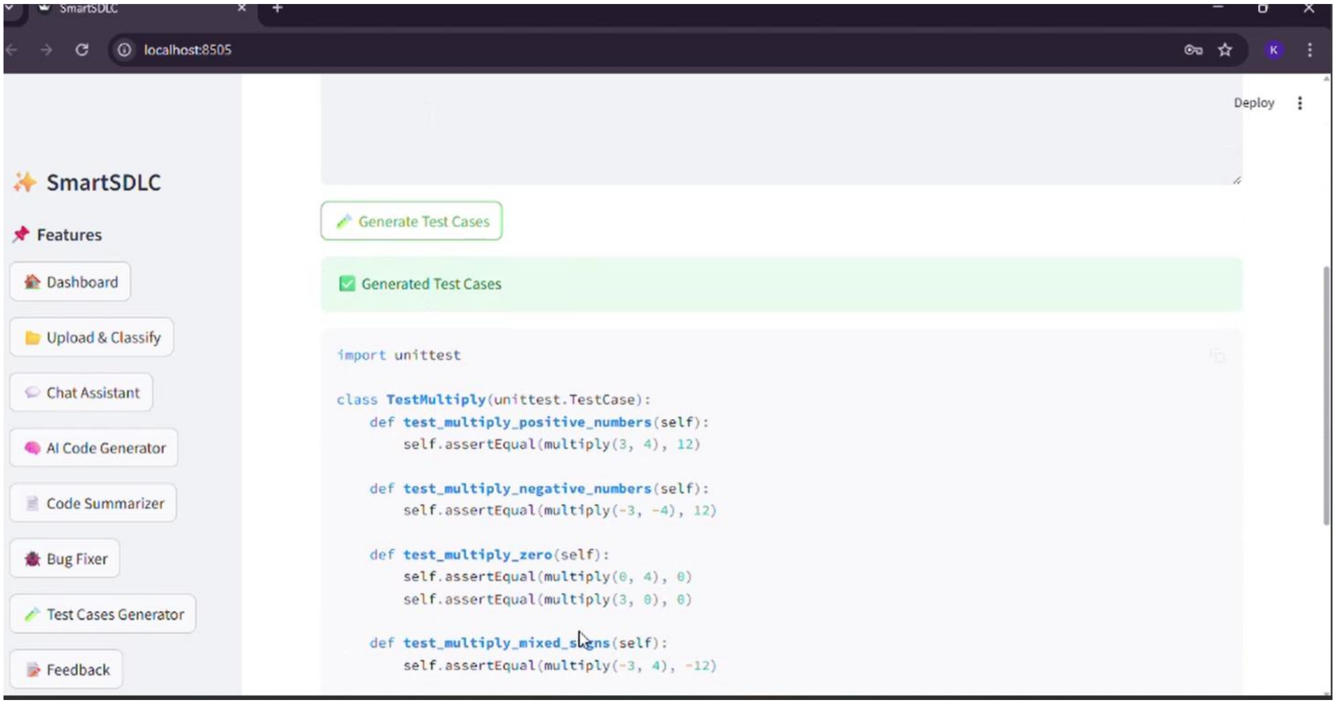
```
def greet(name):
    return f"Hello,{name}!"
```

Summarize Code

Summary

Explanation

greet("Alice").



✅ Generated Test Cases

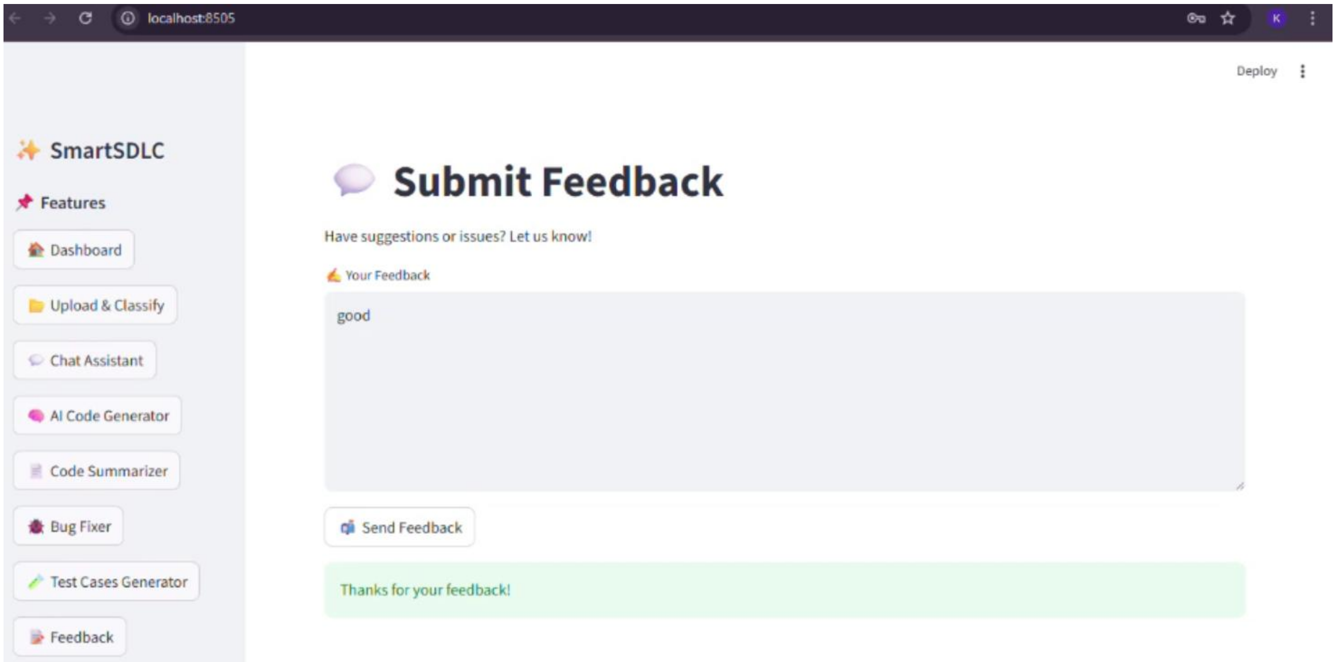
```
import unittest

class TestMultiply(unittest.TestCase):
    def test_multiply_positive_numbers(self):
        self.assertEqual(multiply(3, 4), 12)

    def test_multiply_negative_numbers(self):
        self.assertEqual(multiply(-3, -4), 12)

    def test_multiply_zero(self):
        self.assertEqual(multiply(0, 4), 0)
        self.assertEqual(multiply(3, 0), 0)

    def test_multiply_mixed_signs(self):
        self.assertEqual(multiply(-3, 4), -12)
```



💬 Submit Feedback

Have suggestions or issues? Let us know!

✍ Your Feedback

good

Thanks for your feedback!

8. ADVANTAGES, CONCLUSION & FUTURE SCOPE

8.1 Advantages

1. Increased Development Speed: AI automates repetitive tasks (e.g., code generation, test case creation). Faster requirement analysis through NLP-powered tools. Reduced manual effort across all SDLC phases.

2. Improved Accuracy & Quality: AI detects bugs and vulnerabilities early using static code analysis and pattern recognition. Intelligent test automation ensures higher test coverage and reliability. Predictive analytics help prevent failures and defects.

3. Better Decision Making: AI provides data-driven insights for architecture, design, and technology choices. Risk prediction and mitigation powered by historical data analysis.

4. Automated Documentation & Summarization: Automatically generates and maintains technical and project documentation. Summarizes large requirements or codebases for easier understanding.

5. Continuous Improvement: Machine learning models learn from past projects to improve future development. Feedback loops refine AI models and development practices over time.

6. Enhanced Requirement Analysis: NLP models convert user stories or raw text into structured requirements. Improved traceability and gap analysis between requirements and implementation.

7. Intelligent Assistants: AI-powered coding assistants (like ChatGPT) help with logic building, debugging, and optimization.AI chatbots clarify requirements and support development and testing teams.

8. Seamless Integration Across Tools: SmartSDLC can integrate with Git, JIRA, Jenkins, etc., to automate version control, CI/CD, and issue tracking.AI connects different phases into a cohesive, traceable flow.

9. Reduced Cost and Human Errors: Automation reduces the need for large teams and minimizes human mistakes. Early error detection cuts down costly rework.

10. Scalable and Adaptive: Works well for small to enterprise-level projects. Easily adapts to agile, DevOps, or hybrid methodologies.

8.2 Disadvantages

1. High Initial Cost: Implementing AI tools and training models can be expensive. Requires investment in infrastructure, tools, and skilled personnel.

2. Requires Specialized Skills: Developers, testers, and project managers need AI/ML knowledge. Steep learning curve for teams unfamiliar with AI technologies.

3. Tool and Model Limitations: AI models can produce inaccurate results or biased outputs.Not all project types benefit equally from AI integration.

4. Lack of Explainability: AI decisions (e.g., why a bug was flagged or why a design was suggested) can be difficult to interpret. This “black-box” nature can reduce trust and transparency.

5. Integration Challenges: Integrating AI with existing legacy systems or traditional workflows can be complex. Compatibility issues may arise with current tools or platforms.

6. Data Privacy & Security Risks: AI systems often require access to sensitive project and user data. Improper handling may lead to data breaches or compliance violations.

7. Over-dependence on Automation: Relying too heavily on AI may reduce human creativity and critical thinking. Can lead to neglect of edge cases or intuitive solutions.

8. Testing and Validation Complexity: AI-driven features must be tested rigorously to ensure correctness.AI itself adds another layer that needs validation.

9. Model Training Time and Maintenance: Machine learning models require time and data to train properly. Continuous updates and monitoring are needed to maintain accuracy.

10. Not Suitable for All Projects: For small or short-term projects, AI overhead may not be justified.

Simple workflows may not need SmartSDLC’s advanced capabilities.

9. Conclusion

SmartSDLC represents a transformative shift in how software is developed, tested, and delivered. By integrating Artificial Intelligence into each phase of the SDLC, it significantly boosts efficiency, quality, and decision-making. AI helps automate repetitive tasks, reduce errors, and provide intelligent insights that accelerate development and enhance product reliability. However, SmartSDLC is not without its challenges. It requires significant upfront investment, specialized skills, and careful management of data privacy and AI transparency. It may not be ideal for small or simple projects due to the complexity and cost involved. In conclusion, SmartSDLC is a powerful approach for modern, large-scale, and dynamic software projects, where automation, speed, and intelligence are critical. When implemented correctly, it can lead to smarter, faster, and more adaptive software development processes—making it a strategic advantage in today’s tech-driven world.

10. Future Scope

The future of SmartSDLC is both promising and dynamic, as AI continues to evolve and reshape software development. Here are key areas where SmartSDLC is expected to grow and revolutionize the industry:
1. Fully Autonomous Development Pipelines: End-to-end automation where AI handles requirement gathering, code generation, testing, deployment, and maintenance. "Zero-touch" DevOps with minimal human intervention.
2. Advanced AI-Powered Requirement Engineering: Voice-based or natural language input converted directly into functional requirements or user stories. Real-time collaboration with AI for refining and validating requirements.
3. Generative AI for Code, UI, and Architecture:AI generating optimized, modular, and secure code based on best practices. Automatic design and prototyping using AI-driven design tools.
4. Smart Collaboration Assistants: AI agents supporting teams across time zones with live documentation, suggestions, and risk alerts.AI copilots managing task assignments, sprint planning, and resource optimization.

5. Predictive Analytics for Project Management: AI predicting delays, cost overruns, or team burnout in advance. Dynamic reallocation of tasks/resources based on real-time data.

11. APPENDIX

11.1 Git hub link

https://github.com/sripavani143/SmartSDLC-AI-enhanced-software-development-life-cycle

Demo link:

https://drive.google.com/file/d/1tSQIEdxvnE5YBwQlZvaXsZm7lchnsQMD/view?usp=drivesdk

11.2 Source Code(app.py):

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app import config
from app.routes import ai_routes
app = FastAPI(
    title="SmartSDLC",
    version="1.0",
    description="AI-powered SDLC automation with IBM Watsonx"
)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
app.include_router(ai_routes.router)

@app.get("/")
def read_root():
    return {"message": "SmartSDLC backend is running!"}

from app.routes import auth_routes
app.include_router(auth_routes.router)
```