

# Project Documentation

## 1. Introduction

- Project Title: SmartSDLC - AI-Enhanced Software Development Lifecycle
- Team Members:
  - Karnam Madhuri
  - Inkollu Sushma Meghana
  - Nagamani Pinniboina

## 2. Project Overview

- Purpose: SmartSDLC is an AI-driven platform designed to significantly enhance efficiency and quality across the Software Development Lifecycle. Its purpose is to automate repetitive tasks, provide intelligent assistance, and streamline workflows for software development professionals, leading to faster delivery of higher-quality software.
- Features:
  - Intelligent Requirement Classification & User Story Generation
  - AI Code Generation
  - Automated Bug Fixing
  - Smart Test Case Generation
  - Code Summarization
  - Interactive AI Chatbot Assistant

## 3. Architecture

- Frontend: The frontend is a web-based application built using standard web technologies. It provides a responsive and intuitive user interface with distinct tabs for each AI-powered functionality and a floating chatbot.
  - Technology: HTML, CSS, JavaScript.
- Backend: The backend is a Python-based RESTful API that serves as the central hub for all AI logic and data processing. It receives requests from the frontend, manages the AI model, orchestrates AI functionalities, and handles PDF text extraction.
  - Technology: Python (FastAPI).
- AI Model/Service: The core intelligence is provided by a large language model.
  - Technology: ibm-granite/granite-3.3-2b-instruct (accessed via Hugging Face Transformers library).
- PDF Processing: Integrated within the backend for document analysis.
  - Technology: PyMuPDF

## 4. Setup Instructions

- Prerequisites:
  1. Python 3.8+
  2. Pip (Python package installer)

3. git (for cloning the repository)
  4. A web browser for the frontend (Chrome, Firefox, Edge, Safari recommended)
  5. Google Colab environment (for backend execution, especially for GPU access)
  6. Hugging Face Access Token (for model access)
- Installation:
    1. Clone the repository:  
<https://github.com/Madhuri-Karnam/SmartSDLC---AI-Enhanced-Software-Development-Lifecycle>
    2. Navigate to the backend directory: `cd Project Files/main.py`
    3. Install dependencies: `pip install fastapi uvicorn "python-multipart" "pypdfium2>=4.22.0" "PyMuPDF==1.23.8" "transformers==4.30.2" "torch==2.0.1" "accelerate==0.21.0" "langchain==0.0.240" "langchain-community==0.0.1" "protobuf==3.20.0"`
    4. Set your Hugging Face Token: Ensure your `HUGGING_FACE_TOKEN` is correctly set in `main.py` or as an environment variable.

## 5. Folder Structure

- Client (Frontend):
  - `index.html` : Main application entry point.
  - `css/` (inline in `index.html`): Tailwind CSS configuration and custom styles.
  - `js/` (inline in `index.html`): JavaScript logic for UI interaction and API calls.
- Server (Backend):
  - `main.py`: FastAPI application, AI model integration, API endpoints.
  - `requirements.txt`: List of Python dependencies.

## 6. Running the Application

- Frontend: Open the `index.html` file directly in your web browser. Ensure the `FASTAPI_BASE_URL` in `index.html` points to your running backend.
- Backend: Run the `uvicorn` command as specified in the installation steps within your Google Colab environment or local Python environment.

## 7. API Documentation

- `/classify-requirements`
  - Method: POST
  - Input: `pdf_file` (File, multipart/form-data)
  - Output: JSON object with `classified_data`(categorized requirements).
- `/generate-code`
  - Method: POST
  - Input: `prompt`(Form string)
  - Output: JSON object with `code`(generated code string).
- `/fix-bug`
  - Method: POST

- Input: code\_snippet (Form string)
  - Output: JSON object with original\_code and fixed\_code (strings).
- /generate-test-cases
- Method: POST
  - Input: code\_or\_req(Form string)
  - Output: JSON object with test\_cases (generated test cases string).
- /summarize-code
- Method: POST
  - Input: code\_snippet (Form string)
  - Output: JSON object with summary (code summary string)
- /chatbot
- Method: POST
  - Input: user\_message(Form string)
  - Output: JSON object with ai\_response(chatbot's response string).

## 8. Authentication

- The current prototype of SmartSDLC does not implement user authentication or authorization.
- Note: For a production environment, robust authentication and authorization mechanisms would be critical to secure endpoints and manage user access. Access to the Hugging Face model is managed via an API token within the backend.

## 9. User Interface

### 1. Intelligent Requirement Classification:

Input:

# SmartSDLC

AI-Enhanced Software Development Lifecycle

Requirements
Code Generator
Bug Fixer
Test Case Generator
Code Summarizer

### Requirement Upload & Classification

Upload a PDF document to classify requirements into SDLC phases and generate structured user stories.

Upload PDF Document:

Choose File
Untitled document.pdf

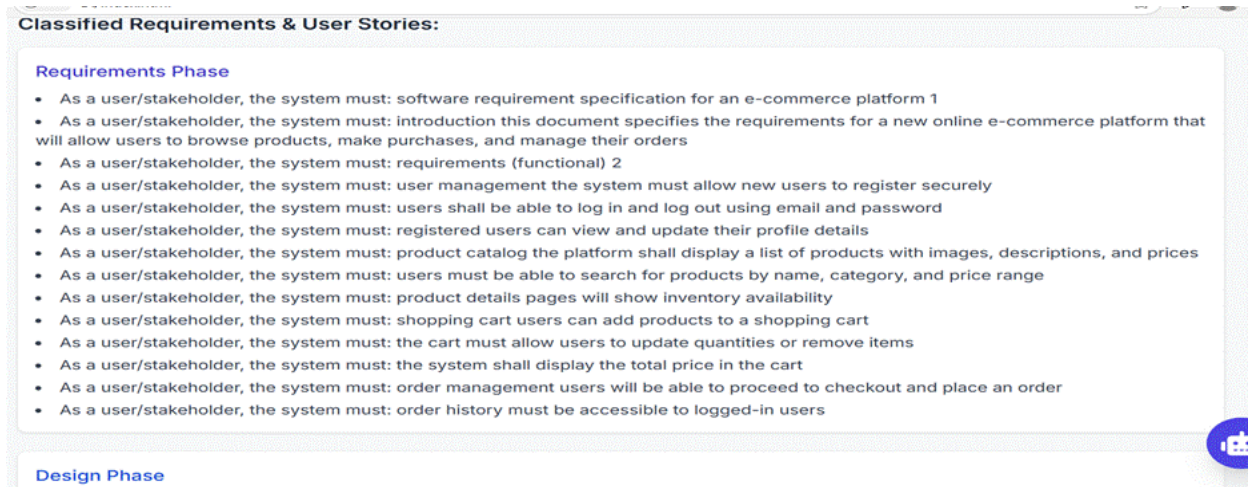
Classify Requirements

Classified Requirements & User Stories:

**Description:** This feature allows users to upload unstructured text documents (primarily PDFs) containing project requirements. The system then processes this text, identifies key requirement

sentences, and classifies them into standard SDLC phases (e.g., Requirements, Design, Development, Testing, Deployment, Other). It can also generate structured user stories from the extracted information.

## Output:



**Classified Requirements & User Stories:**

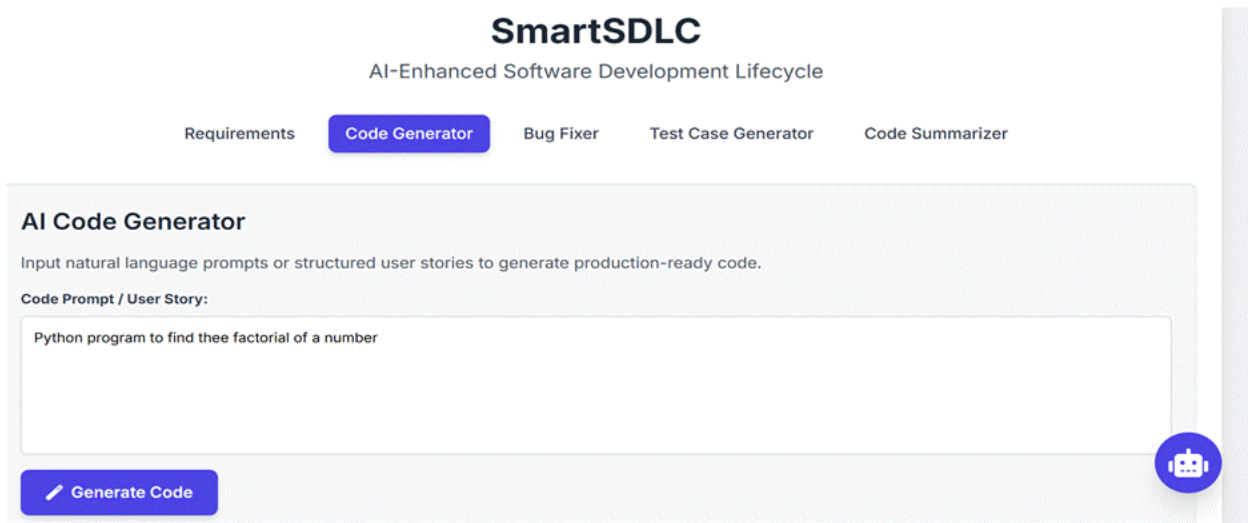
**Requirements Phase**

- As a user/stakeholder, the system must: software requirement specification for an e-commerce platform 1
- As a user/stakeholder, the system must: introduction this document specifies the requirements for a new online e-commerce platform that will allow users to browse products, make purchases, and manage their orders
- As a user/stakeholder, the system must: requirements (functional) 2
- As a user/stakeholder, the system must: user management the system must allow new users to register securely
- As a user/stakeholder, the system must: users shall be able to log in and log out using email and password
- As a user/stakeholder, the system must: registered users can view and update their profile details
- As a user/stakeholder, the system must: product catalog the platform shall display a list of products with images, descriptions, and prices
- As a user/stakeholder, the system must: users must be able to search for products by name, category, and price range
- As a user/stakeholder, the system must: product details pages will show inventory availability
- As a user/stakeholder, the system must: shopping cart users can add products to a shopping cart
- As a user/stakeholder, the system must: the cart must allow users to update quantities or remove items
- As a user/stakeholder, the system must: the system shall display the total price in the cart
- As a user/stakeholder, the system must: order management users will be able to proceed to checkout and place an order
- As a user/stakeholder, the system must: order history must be accessible to logged-in users

**Design Phase**

## 2. AI Code Generator:

### Input:



**SmartSDLC**  
AI-Enhanced Software Development Lifecycle

Requirements **Code Generator** Bug Fixer Test Case Generator Code Summarizer

**AI Code Generator**

Input natural language prompts or structured user stories to generate production-ready code.

Code Prompt / User Story:

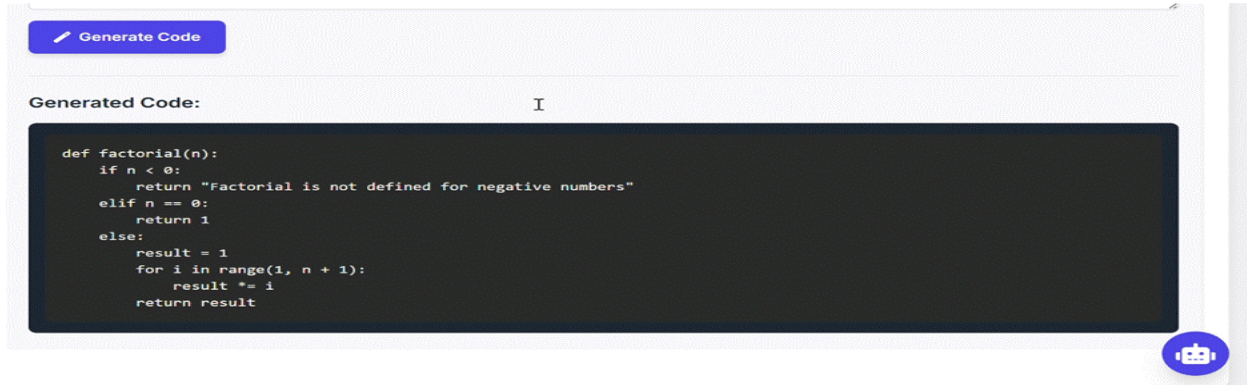
Python program to find the factorial of a number

**Generate Code**

**Description:** This powerful tool enables developers to generate functional code snippets or full functions by simply providing a natural language prompt or a user story. It significantly accelerates the coding process for boilerplate, common algorithms, or initial feature implementations.



Output:



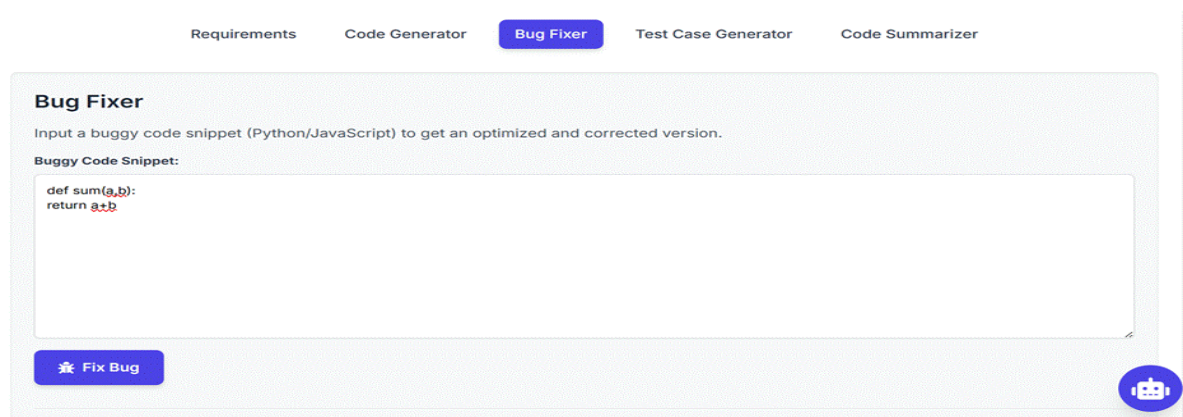
The screenshot shows a web interface with a purple button labeled "Generate Code". Below it, the text "Generated Code:" is followed by a code editor containing the following Python code:

```
def factorial(n):  
    if n < 0:  
        return "Factorial is not defined for negative numbers"  
    elif n == 0:  
        return 1  
    else:  
        result = 1  
        for i in range(1, n + 1):  
            result *= i  
        return result
```

A small blue icon with a robot face is visible in the bottom right corner of the interface.

### 3. Automated Bug Fixer:

Input:



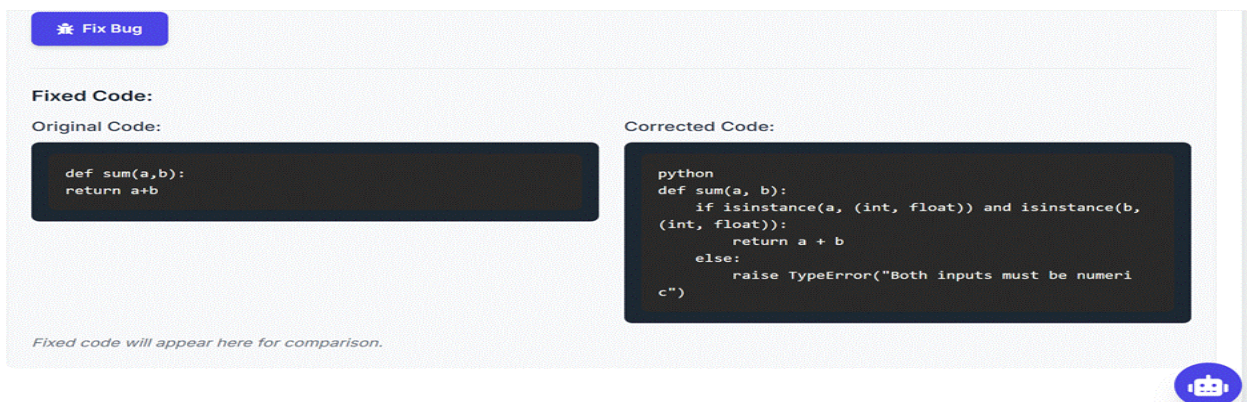
The screenshot shows a web interface with a navigation bar containing "Requirements", "Code Generator", "Bug Fixer" (highlighted), "Test Case Generator", and "Code Summarizer". Below the navigation bar, the "Bug Fixer" section is active. It contains the text "Input a buggy code snippet (Python/JavaScript) to get an optimized and corrected version." and "Buggy Code Snippet:". Below this is a code editor with the following code:

```
def sum(a,b):  
    return a+b
```

A purple button labeled "Fix Bug" is located at the bottom left of the input section. A small blue icon with a robot face is visible in the bottom right corner of the interface.

**Description:** Designed to assist developers in quickly identifying and correcting errors. Users can paste a buggy code snippet, and the AI analyzes it to suggest and provide an optimized, corrected version, reducing manual debugging time and improving code quality.

Output:



The screenshot shows the "Bug Fixer" output section. It features a purple button labeled "Fix Bug" at the top left. Below it, the text "Fixed Code:" is followed by two code editors. The left editor, labeled "Original Code:", contains the following code:

```
def sum(a,b):  
    return a+b
```

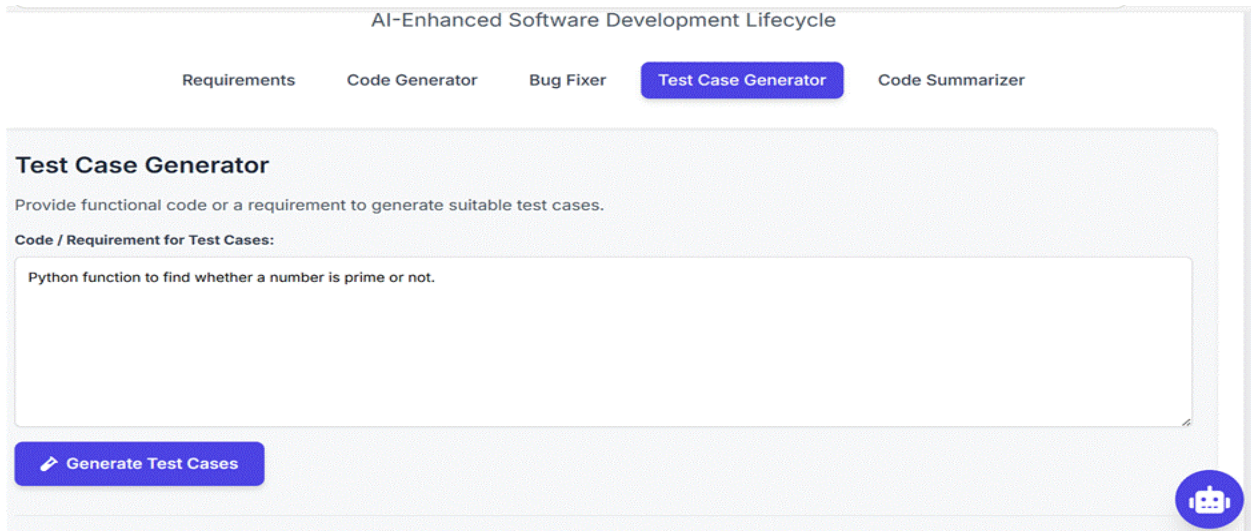
The right editor, labeled "Corrected Code:", contains the following code:

```
python  
def sum(a, b):  
    if isinstance(a, (int, float)) and isinstance(b, (int, float)):  
        return a + b  
    else:  
        raise TypeError("Both inputs must be numeri  
c")
```

At the bottom left, there is a note: "Fixed code will appear here for comparison." A small blue icon with a robot face is visible in the bottom right corner of the interface.

## 4. Smart Test Case Generator:

### Input:



The screenshot shows a web application titled "AI-Enhanced Software Development Lifecycle". It has a navigation bar with five tabs: "Requirements", "Code Generator", "Bug Fixer", "Test Case Generator" (which is highlighted in blue), and "Code Summarizer". Below the navigation bar, the "Test Case Generator" section is displayed. It contains a heading "Test Case Generator" and a subtext "Provide functional code or a requirement to generate suitable test cases." Below this is a text input field with the placeholder text "Code / Requirement for Test Cases:". The input field contains the text "Python function to find whether a number is prime or not." Below the input field is a blue button with a white icon of a document and the text "Generate Test Cases". In the bottom right corner of the interface, there is a blue circular icon with a white robot head.

**Description:** This feature automates the creation of test cases. Developers or QA engineers can provide either functional code or a software requirement, and the AI will generate relevant unit or integration test cases, often adhering to specific testing frameworks like unittest or pytest.

### Output:



The screenshot shows the output of the "Test Case Generator" feature. It displays a Python code snippet for a unittest class named "TestIsPrime". The code is as follows:

```
class TestIsPrime(unittest.TestCase):

    def test_is_prime_positive_integers(self):
        self.assertTrue(is_prime(2))
        self.assertTrue(is_prime(3))
        self.assertTrue(is_prime(5))
        self.assertTrue(is_prime(7))
        self.assertTrue(is_prime(11))
        self.assertFalse(is_prime(4))
        self.assertFalse(is_prime(6))
        self.assertFalse(is_prime(8))
        self.assertFalse(is_prime(9))
        self.assertFalse(is_prime(10))

    def test_is_prime_zero_and_negative(self):
        self.assertFalse(is_prime(0))
        self.assertFalse(is_prime(-1))
        self.assertFalse(is_prime(-2))

if __name__ == '__main__':
    unittest.main()
```

In the bottom right corner of the interface, there is a blue circular icon with a white robot head.



## 5. Code Summarizer:

Input:

### SmartSDLC

AI-Enhanced Software Development Lifecycle

RequirementsCode GeneratorBug FixerTest Case GeneratorCode Summarizer

#### Code Summarizer

Input any source code snippet or module to get a human-readable explanation.

Code Snippet to Summarize:

```
def fibonacci_iterative(n_terms):  
    """  
    Generates the Fibonacci series up to n_terms using an iterative approach.  
    """  
    if n_terms <= 0:  
        print("Please enter a positive integer.")  
        return  
    elif n_terms == 1:  
        print("Fibonacci Series:")  
        print(0)  
        return
```

Summarize Code

**Description:** To enhance code comprehension and knowledge transfer, this tool provides human-readable explanations and summaries of any given code snippet or module. It's invaluable for new team members, code reviews, or understanding complex legacy codebases.

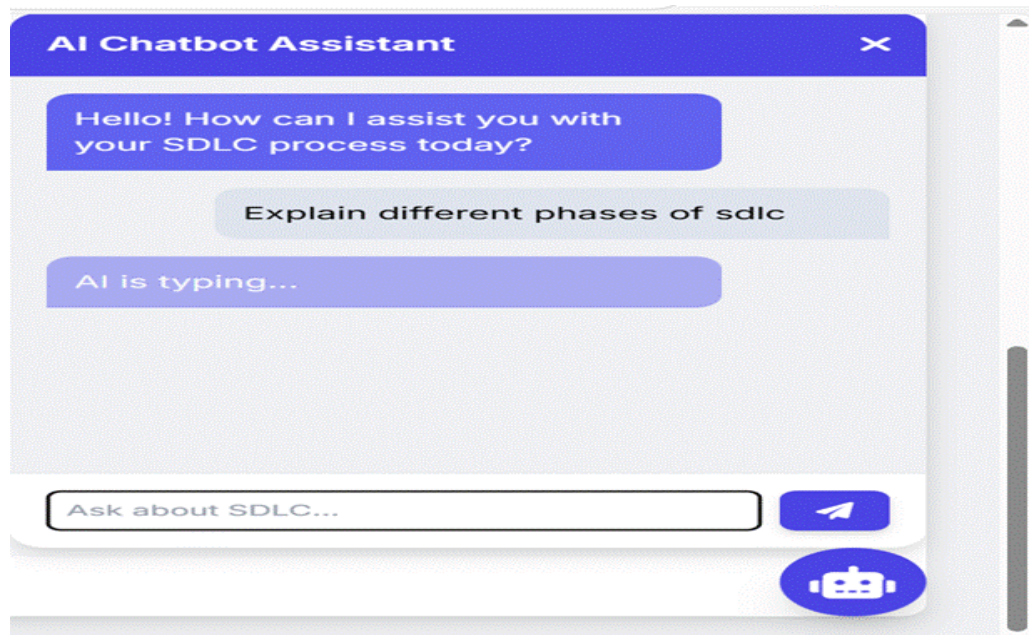
Output:

#### Code Summary:

The provided Python function 'fibonacci\_iterative(n\_terms)' generates the first 'n\_terms' of the Fibonacci series using an iterative method. The Fibonacci series is a sequence of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1. This function checks for valid input (positive integers only), prints the initial terms manually, and then uses a loop to calculate subsequent terms until it reaches the desired count ('n\_terms'). It's useful for generating the first few terms of the Fibonacci series without computing all terms upfront, which can be efficient for larger values of 'n'. Explanation: 1. Function definition: - The function takes one argument, 'n\_terms', which specifies the number of Fibonacci terms to generate. 2. Input validation: - It checks if 'n\_terms' is less than or equal to zero; if so, it prompts the user for a positive integer and returns early. - If 'n\_tabs' equals 1, it prints the first term directly as 0 and terminates. 3. Initialization: - Variables 'a' and 'b' are initialized to the first two Fibonacci numbers, 0 and 1 respectively. - The first two terms are printed explicitly before entering the loop. 4. Iteration and calculation: - A for-loop runs from 2 to 'n\_terms', calculating each term by adding the previous two ('a' and 'b'), storing the result in 'next\_term', and updating 'a' and 'b' accordingly. - Inside the loop, each calculated term is printed followed by a space. 5. Output formatting: - After printing all terms, the function prints a newline character ('print()' with no arguments) to move to the next line in the output. Use Cases: - Education: Demonstrating the Fibonacci series concept and iteration. - Coding practice: Helping beginners understand loops, variable initialization, and function handling in Python. - Applications requiring Fibonacci sequence generation: Such as algorithms, simulations, or problem-solving tasks that involve Fibonacci sequences.

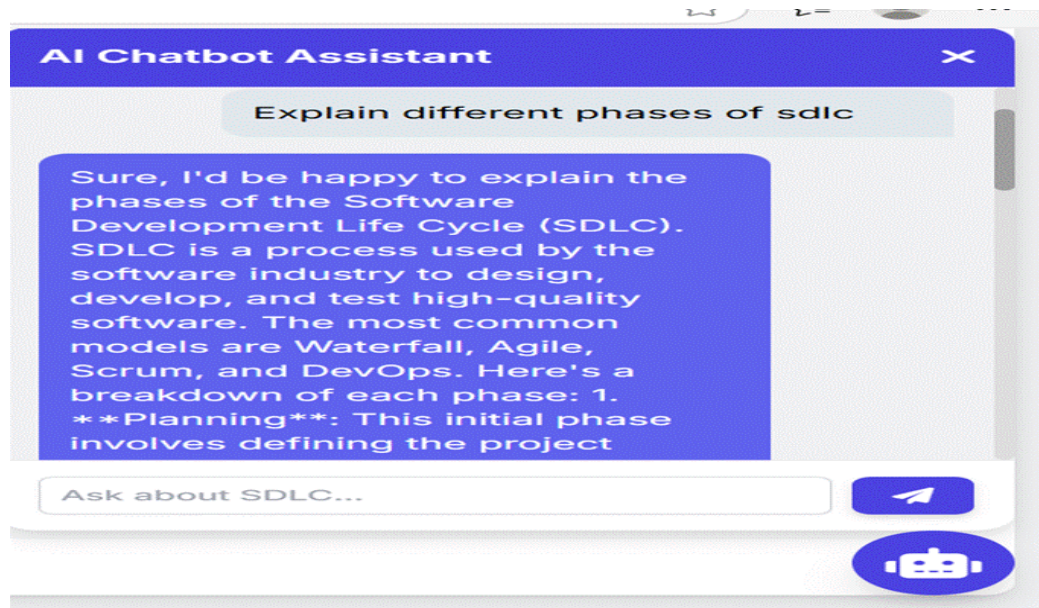
## 6. Interactive AI Chatbot Assistant:

Input:



**Description:** A conversational AI assistant available on-demand to answer questions related to the Software Development Lifecycle. It can provide information on methodologies (Agile, Scrum), best practices, specific tools, or general programming concepts, acting as an instant knowledge base.

Output:





## 10. Testing

- **Testing Strategy:** The project utilized a combination of functional and performance testing. Functional tests verified the correctness of AI-generated outputs and core feature functionalities (e.g., PDF parsing, code generation accuracy). Performance tests focused on API response times and stability under load.
- **Tools Used:** Manual testing for UI/UX. Python's requests library for API testing. Basic timing mechanisms for performance measurement. (For comprehensive testing, frameworks like pytest for backend unit/integration tests and browser automation tools for frontend E2E tests would be used.)

## 11. Screenshots or Demo

- **Project Demo Link:**  
<https://drive.google.com/file/d/1xgeYCrtmhoBJwuW1YLYb5UlvFUFgm0yE/view?usp=sharing>

## 12. Known Issues

- **AI Model Limitations:** The ibm-granite model, while powerful, may occasionally produce "hallucinations" or suboptimal outputs, requiring human review.
- **PDF Parsing Variations:** Complex PDF layouts or scanned images may result in less accurate text extraction.
- **Limited Language Support:** Currently, code generation and analysis primarily focus on Python.
- **No Persistent Storage:** User data and chat history are not persistently stored across sessions in this prototype.
- **No User Authentication:** The prototype lacks user login/registration and role-based access control.

## 13. Future Enhancements

- Implement robust user authentication and authorization.
- Integrate with popular IDEs (e.g., VS Code extensions) for seamless workflow.
- Expand AI capabilities to support more programming languages and advanced tasks (e.g., automated refactoring, design pattern suggestions).
- Integrate with CI/CD pipelines and project management tools (e.g., Jira, GitHub Actions).
- Add persistent storage for user preferences, project data, and chat history.
- Develop a more sophisticated UI/UX with advanced code highlighting and interactive elements.
- Explore fine-tuning the AI model on domain-specific codebases for improved accuracy.
- Implement comprehensive logging and monitoring for production deployments.

