SmartSDLC – AI-Enhanced Software Development Lifecycle Documentation

1.Introduction

• Project Title: SmartSDLC – AI-Enhanced Software Development Lifecycle
• Team Members:

Hema.V

Brindha Lakshmi.A

Dharshini priya.D

2.Project Overview

• Purpose:
SmartSDLC is an AI-powered platform designed to automate and streamline the Software Development Lifecycle (SDLC). It leverages IBM Watsonx Granite models, LangChain, FastAPI, and Streamlit to enhance each phase of software engineering, including requirements analysis, code generation, test case creation, bug fixing, and documentation. By integrating AI-driven automation, SmartSDLC reduces manual workload, accelerates development timelines, and improves software quality.

• Features:
- Requirement Upload & Classification
Key Point: Structured requirement management
Functionality: Extracts text from uploaded PDFs and classifies sentences into SDLC phases (Requirements, Design, Development, Testing, Deployment).

- AI Code Generator
Key Point: Automated code creation
Functionality: Generates clean, production-ready code from natural language prompts or structured user stories.

- Bug Fixer
Key Point: Error detection and correction
Functionality: Identifies and fixes syntax and logic errors in code snippets, returning optimized versions.

- AI-Driven Test Case Generation
  Key Point: Automated testing support
  Functionality: Generates unit and integration test cases from generated or user-provided code.

- Code Summarization & Documentation
  Key Point: Improved maintainability
  Functionality: Summarizes and documents code to improve readability and project understanding.

- Chatbot Assistance
  Key Point: Real-time developer support
  Functionality: Provides interactive guidance and assistance for SDLC-related queries.

- GitHub Integration
  Key Point: Workflow automation
  Functionality: Automates pushing code, opening issues, and syncing documentation with GitHub repositories.

• Use Case Scenarios:
- Requirement Upload & Classification: Upload raw requirement PDFs and receive structured user stories grouped by SDLC phase.
- AI Code Generator: Generate working Python or JavaScript code from natural language prompts.
- Bug Fixer: Submit buggy code and receive AI-optimized corrections.
- Test Case Generation: Automatically generate test cases for faster validation.
- Chatbot Support: Access an AI-powered assistant to answer SDLC queries and guide workflows.

3. Architecture

- Frontend (Streamlit): Interactive dashboard for requirements, code generation, bug fixing, testing, and chatbot interaction.
- Backend (FastAPI): API layer handling routing, authentication, AI requests, and service orchestration.
- AI Integration (IBM Watsonx + LangChain): Natural language processing, code generation, bug fixing, and summarization.
- Modules: Requirement analysis, code generation, bug fixing, test case generation, code summarization, GitHub workflows.
- Deployment: Local hosting with Uvicorn (backend) and Streamlit (frontend).

4.Setup Instructions

Prerequisites:

- Python 3.10+
- FastAPI, Uvicorn
- Streamlit
- IBM Watsonx API access
- LangChain
- PyMuPDF (fitz)
- Git & GitHub

Installation Process:
- Install Python 3.10 and pip
- Create virtual environment: python -m venv myenv
- Activate environment and install dependencies from requirements.txt
- Configure .env file with API keys and model IDs
- Start FastAPI backend: uvicorn app.main:app --reload
- Run Streamlit frontend: streamlit run frontend/Home.py

5. Folder Structure

- app/ – FastAPI backend
  - routes/ – API endpoints for AI, chat, auth, feedback
  - services/ – Core AI service logic
  - models/, utils/ – Supporting modules
- frontend/ – Streamlit UI components
  - Home.py – Entry dashboard
  - pages/ – Modular pages (requirements, code gen, bug fixer, etc.)
- ai_story_generator.py – Requirement classification
- code_generator.py – Code and test case generation
- bug_resolver.py – Bug fixing
- doc_generator.py – Code summarization
- conversation_handler.py – Chatbot logic
- github_service.py – GitHub workflow automation

6. Running the Application

➢Start FastAPI backend with Uvicorn
➢Run Streamlit dashboard
➢Navigate via dashboard menu
➢Upload requirements or enter prompts
➢Generate code, fix bugs, create tests, and access chatbot
➢Sync outputs with GitHub and export documentation

7. API Documentation

- POST /upload-pdf – Uploads requirements for classification

- POST /generate-code – Generates production-ready code
- POST /fix-bugs – Accepts buggy code and returns corrected version
- POST /generate-tests – Creates test cases
- POST /summarize-code – Summarizes uploaded code
- POST /chat – Chatbot interactions
- POST /feedback – Submits user feedback
- GET /docs – Swagger UI for API exploration

8. Authentication

- Token-based authentication (JWT)
- Role-based access (admin, developer, tester)
- Hashed user login and registration
- Planned: OAuth2 integration and session management

9. User Interface

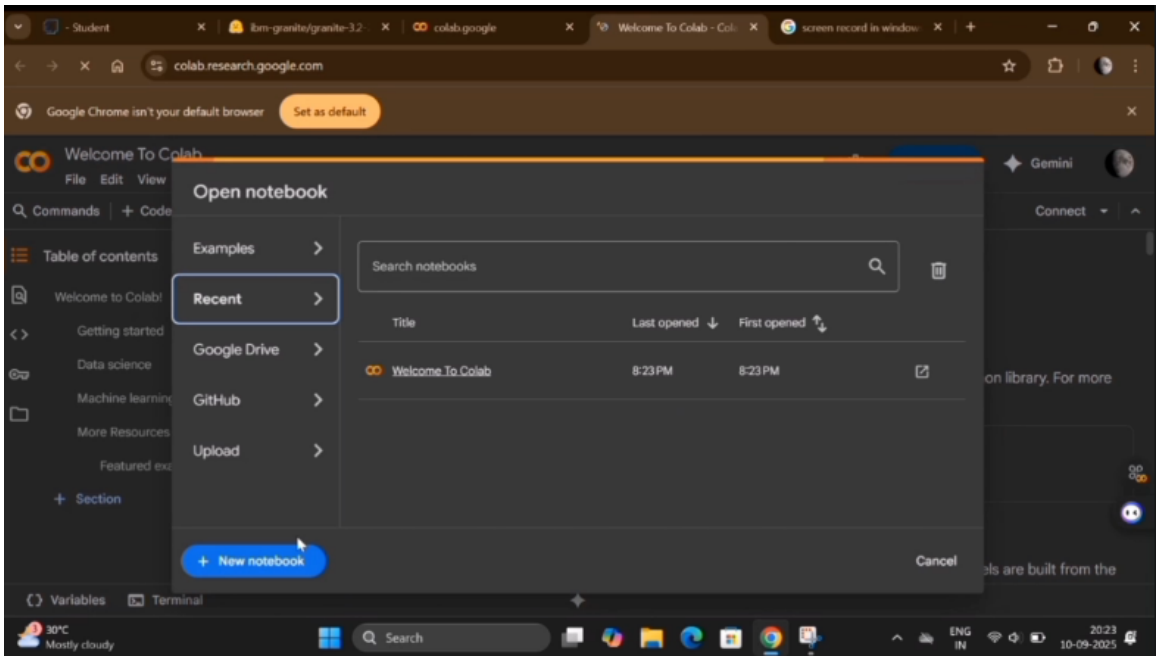
- Home Dashboard: Feature overview and navigation
- Requirement Classifier: Upload and classify requirements
- Code Generator: Prompt-based code creation
- Bug Fixer: Code correction interface
- Test Generator: Auto-generated test cases
- Chatbot: Real-time AI guidance
- Feedback Form: Collects user feedback
- GitHub Sync: Push code, open issues, sync docs

10. Testing

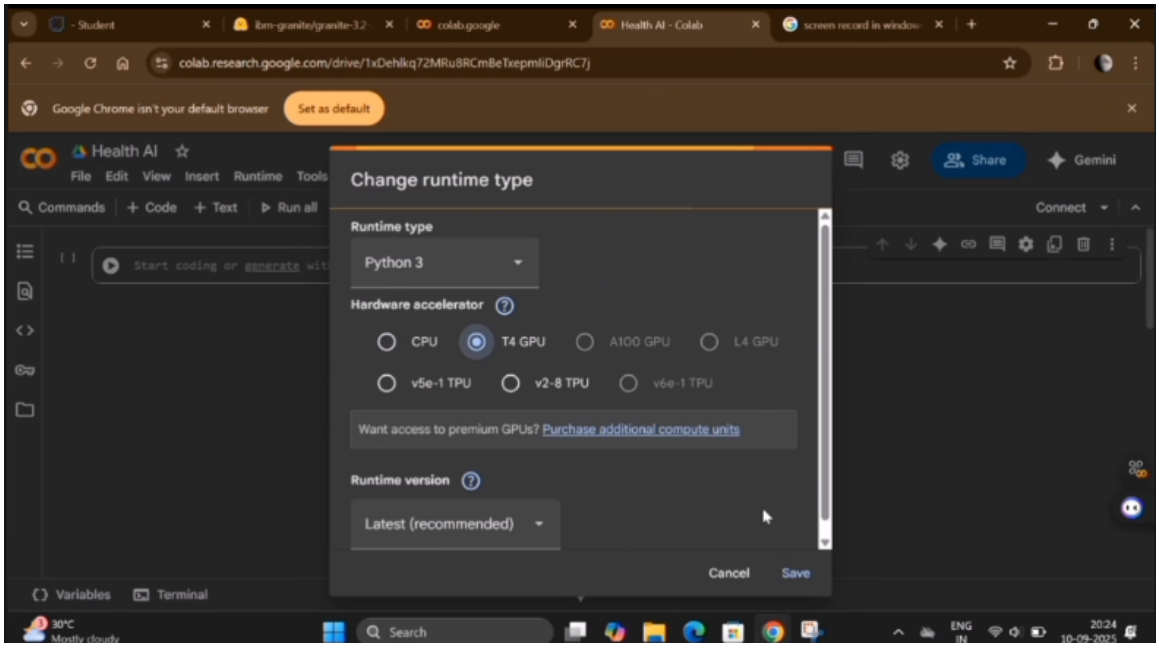
- Unit Testing: For backend AI services
- API Testing: Swagger UI and Postman
- Manual Testing: For requirement classification, bug fixing, and chatbot
- Edge Cases: Large PDF uploads, malformed prompts, incorrect API keys

11. Screenshots

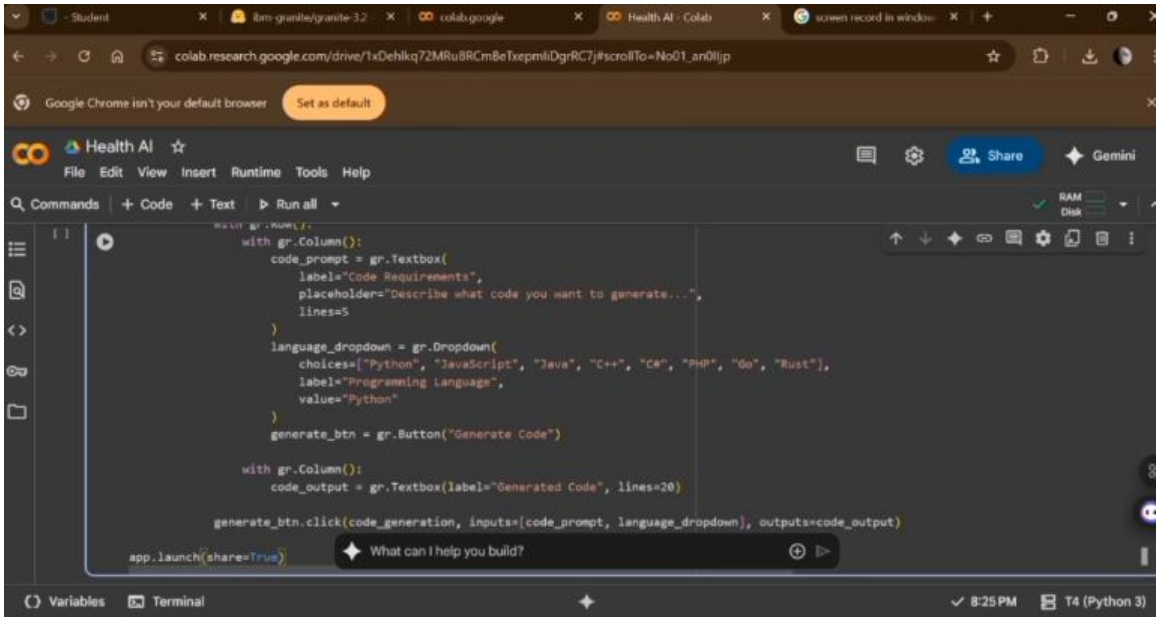
[Insert mockups: Home Dashboard, Requirement Upload, AI Code Generator, Bug Fixer, Test Generator, Chatbot, GitHub Sync]



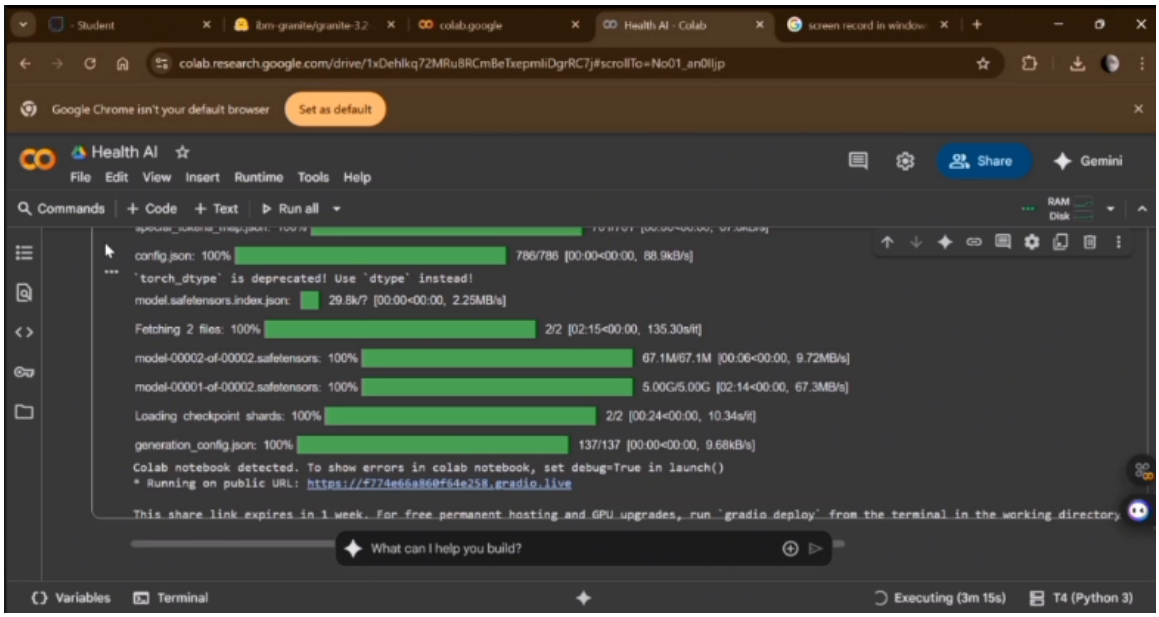
Search for “Google Colab” in any browser.



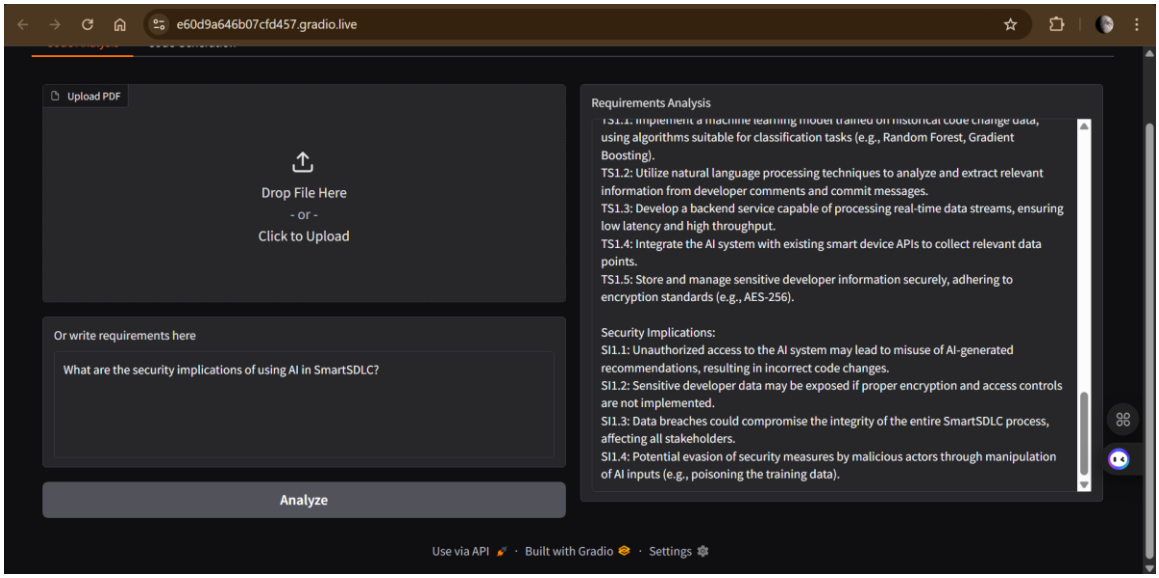
Change the title of the notebook”Untitled “ to “Health AI”.Then click on “Runtime”,then go to “Change Runtime Type.Choose “T4 GPU” and click on “Save”.



Then run the code in the cell.



Click on the URL to open the Gradio Application click on the link.



Give the requirements and analyse.Thus the output will be generated.

12. Known Issues

- Limited offline support
- Dependency on IBM Watsonx API availability
- Occasional latency for large PDFs
- Basic test case generation (needs extension)

13. Future Enhancements

- CI/CD pipeline integration
- Multi-language support for code generation
- Advanced bug detection with deep learning
- Cloud deployment (AWS, IBM Cloud, Azure)
- Collaboration features for team workflows
- Enhanced test generation with coverage analysis