Smart SDLC – AI Enhanced Software Development Lifecycle

## Introduction

**Project Title**: Smart SDLC

## Team Members and their Roles:

* 1. **B. Bogendra Babu (Project Lead, Full Stack Developer, AI/NLP Specialist):**
     + **Designed overall architecture, developed Stream-lit front-end for modules.**
     + **Coordinated AI integration and back-end planning.**
     + **Final integration of all SDLC modules.**
     + **Integrated Open-AI/Gemini API s for chat-bot, implemented AI for requirement generation, code, and test cases.**
     + **Planned WatsonX/Granite integration.**
  2. **B. Anvesh Babu (Frontend Developer):**
     + **Designed UI for Login, Dashboard, and Wizard pages.**
     + **Implemented dark/light theme toggle.**
     + **Assisted with navigation and page routing.**
  3. **B. Khyathi Priyanshu (Authentication & Database Developer):**
     + **Developed login using Stream-lit.**
     + **Set up SQLite database for users, chat logs, requirements, and feedback.**
     + **Managed schema creation and storage logic.**
  4. **B. Chinnama Naidu (Data & Feedback Analyst):**
     + **Implemented sentiment analysis using Text-blob.**
     + **Designed feedback form and analytic dashboard.**
     + **Created Plot visualizations for feedback trends.**

1. **Project Overview**

### Purpose:

Smart SDLC is a generative AI-powered application developed using Fast-API and IBM's Granite-3.3-2b-instruct model. It provides phase-wise assistance across the Software Development Life-cycle (SDLC), enabling developers and learners to understand and implement software engineering practices through intelligent automation.

### Features:

Phase-wise AI chatbot (Gradio-powered) Generates code, test cases, design suggestions Secure login and admin dashboard

Sentiment analysis and feedback tracking Local and Colab-based deployment

1. **Architecture Frontend:**

* Stream-lit used for main user interface and SDLC modules.
* Gradio and Jinja2 templates used for chatbot UI and rendering static pages.

### Backend:

* Fast-API server with Restful endpoints handling login, registration, dashboard, chat, and feedback processing.

### Database:

* SQLite used to manage user sessions, chat history, requirements, and sentiment-tracked feedback.

# Setup Instructions

### Prerequisites:

Python 3.10+, FastAPI, Uvicorn, Gradio, SQLite, langchain-ibm, Jinja2

### Installation:

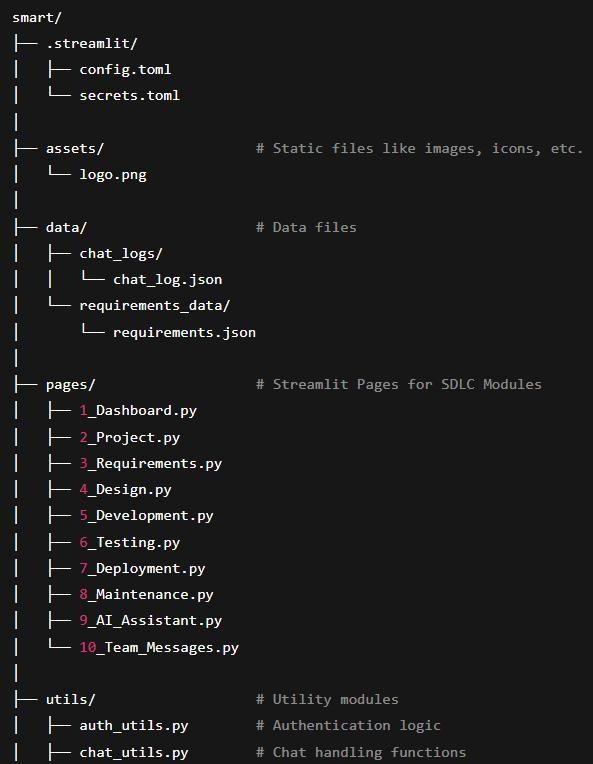
git clone <repo-url> pip install -r requirements.txt uvicorn app:app --reload

# Folder Structure

### Client:

Contains Gradio and Jinja templates in /templates/

### Server:

FastAPI logic, routes, and processing in app.py

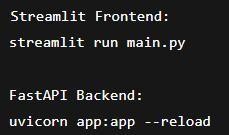
# Running the Application

### Front-end:

Access the app at [http://127.0.0.1:8000](http://127.0.0.1:8000/)

### Back-end:

Start with:

uvicorn app:app --reload

# API Documentation

### Endpoints:

/, /login, /register, /dashboard, /chat, /feedback, /admin

### Details:

Accepts JSON requests

Uses sessions and cookies for tracking

Returns structured responses for UI rendering and analytics

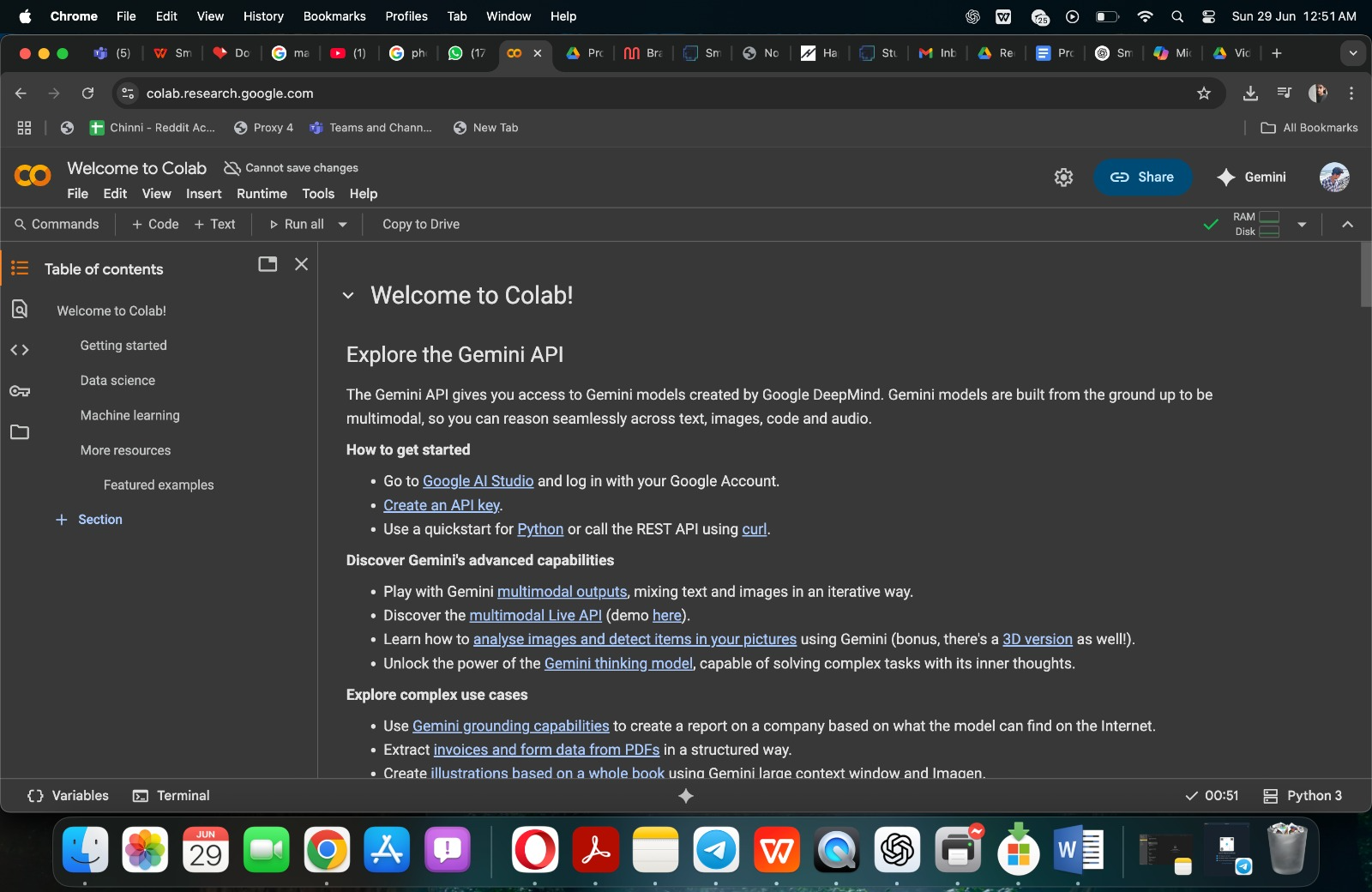
# Authentication

### Security:

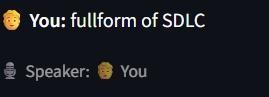
User login/register with session-based token handling Admin dashboard available only to authorized users

# User Interface

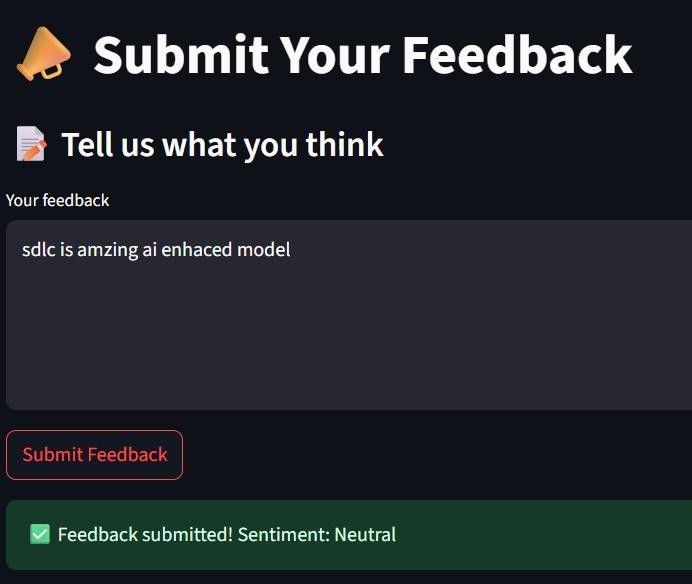
Built using Gradio and Colab templates Admin dashboard includes visual analytics Chat interface shows context-based AI ponses



**Chatbot AI -Assitent:**



**Feedback Analysis:**



# Purpose

Smart SDLC was built to address the inefficiencies and manual overhead in traditional software development lifecycles by embedding AI assistance in each phase. Its purpose is to:

Provide real-time, contextual support using natural language.

Automate repetitive development tasks like writing boilerplate code, creating test cases, and drafting documentation.

Enable learners and non-technical users to grasp SDLC concepts interactively. Enhance consistency and quality of outputs across the development process.

**Traditional SDLC vs Smart SDLC (AI-Powered)**

Aspect Traditional SDLC Smart SDLC (AI-Enhanced)

Requirement Gathering

Manual meetings, user stories

AI generates requirement specifications

Design

Diagrams and documents prepared manually

AI suggests design structures and components

Development

Written entirely by developers

AI provides code templates/snippets

Testing

Test cases manually written AI generates test cases and edge

and run scenarios

Deployment Planned by DevOps/PM

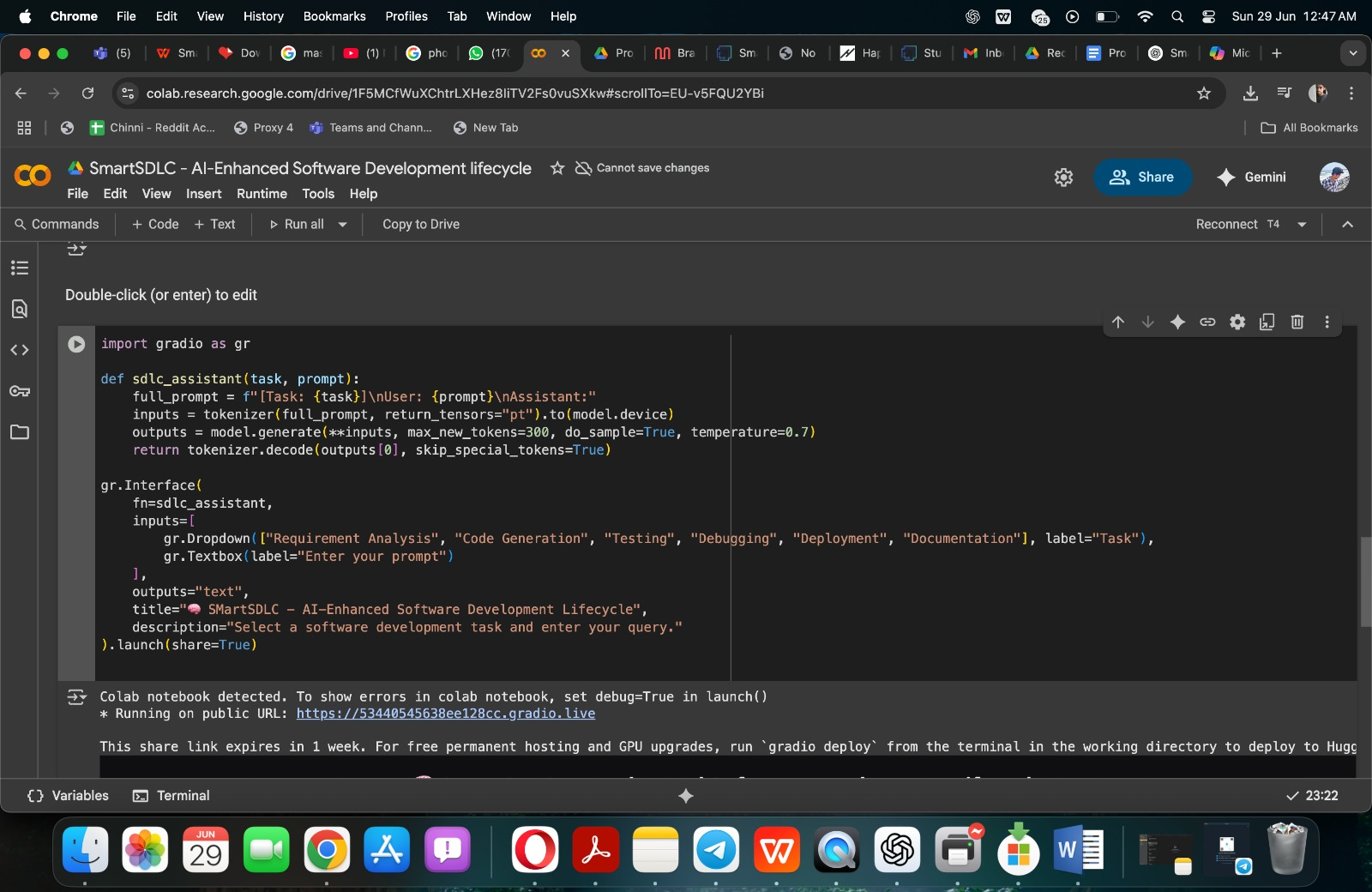
AI recommends deployment strategies/tools

Maintenance

Reactive to bugs and user feedback

AI gives refactoring and optimization guidanc

# Testing



Manual testing for all SDLC phases Sentiment logging and feedback tested

Dummy user data tested through complete flow

# Importance of Smart SDLC

**Time-Saving**: Automates documentation, test case generation, and boilerplate code.

**Educational Value**: Helps students understand phase-wise breakdown and implementation.

**Increased Consistency**: Reduces human errors in documentation and coding. **Prototyping Power**: Startups can quickly ideate, test, and refine product concepts. **Decision Support**: Offers suggestions based on patterns and prior interactions.

1. **Demo link:**

[**https://drive.google.com/file/d/1gNtlSFI1oqrCaWfV1cmkV95Ic3wKRRcN/view?usp=drivesdk**](https://drive.google.com/file/d/1gNtlSFI1oqrCaWfV1cmkV95Ic3wKRRcN/view?usp=drivesdk)

**Reliance on Prompts**: AI output quality heavily depends on how queries are phrased.

**Generalization**: May not generate production-ready solutions for complex enterprise needs.

**No Real-Time Collaboration**: Currently not multi-user or team-oriented.

**Limited Integration**: Doesn't yet support third-party tools like Postman or GitHub Actions.

**Data Privacy**: AI models may not be suitable for use with sensitive or proprietary data.

## Known Issues

-Varying AI response quality with vague prompts.

* + Admin dashboard insights are dependent on sufficient feedback data.
  + Occasional token/session expiry during long idle periods.

## Future Enhancements

Add support for generating architecture diagrams Export project plans and reports in PDF Integration with CI/CD pipelines

API testing and mock server integration

Use LangChain to chain prompts for better AI conversations

Export project reports, charts, and summaries as downloadable PDFs Integrate CI/CD pipelines (e.g., GitHub Actions, Jenkins)

Add Postman-ready APIs and support for OpenAPI 3.0 Add team collaboration with multi-user chat & versioning

Add JWT or OAuth2-based secure token-based authentic