**SMART SDLC - AI ENHANCED SOFTWARE DEVELOPMENT LIFECYCLE**

**Project Documentation**

**1. Introduction**

* **Project title**: SmartSDLC – AI Enhanced Software Development Lifecycle
* **Team member**: PRIYADHARSHINI R
* **Team member**: SANDHYA S
* **Team member**: PRIYADHARSHINI M
* **Team member**: SANGAVI K

**2. Project Overview**

The SmartSDLC project is an AI-enhanced platform designed to automate and streamline the Software Development Lifecycle (SDLC) using advanced technologies like IBM Watsonx, FastAPI, LangChain, and Streamlit. It integrates generative AI to handle key SDLC phases, including requirement analysis, code generation, test case creation, bug fixing, and documentation.

The platform features a user-friendly interface that allows users to upload PDFs, generate structured requirements, and transform natural language prompts into functional code. It also includes an AI-powered chatbot for real-time assistance and support. The backend, built with FastAPI, efficiently processes API requests, while the frontend, developed using Streamlit, offers a visually appealing and interactive dashboard. The system is modular, scalable, and secure, featuring a robust authentication mechanism and seamless integration between AI models and user inputs. Deployment is streamlined with local hosting via Uvicorn and Streamlit, with comprehensive API documentation available through Swagger UI. By leveraging AI in a practical and efficient manner, SmartSDLC significantly reduces manual workload, accelerates development processes, and enhances software quality, making it an invaluable tool for developers and project teams.

*Scenario 1: Requirement Upload and Classification*

Requirement Upload and Classification, the platform simplifies the complex task of requirement gathering by allowing users to upload PDF documents containing raw, unstructured text. The backend extracts content using PyMuPDF and leverages IBM Watsonx’s Granite-20B AI model to classify each sentence into specific SDLC phases such as Requirements, Design, Development, Testing, or Deployment. These classified inputs are then transformed into structured user stories, enabling clear planning and traceability. The frontend displays this output in an organized, readable format grouped by phase, significantly improving clarity and saving manual effort.

*Scenario 2: AI Code Generator*

AI Code Generator, addresses the development phase, where developers can input natural language prompts or structured user stories. These prompts are sent to the Watsonx model, which generates contextually relevant, production-ready code. This reduces the time needed for boilerplate or prototype creation and enhances coding efficiency. The code is presented in a clean, syntax-highlighted format on the frontend, ready for use or further enhancement.

*Scenario 3: Bug Fixer*

Bug Fixer, the platform supports debugging by accepting code snippets in languages such as Python or JavaScript. Upon receiving the buggy code, the Watsonx AI analyzes it for both syntactical and logical errors and returns an optimized version. This not only assists developers in identifying mistakes without extensive manual reviews but also provides immediate, corrected code directly in the frontend for comparison.

* **Purpose**:  
   The purpose of SmartSDLC is to transform the traditional Software Development Lifecycle (SDLC) using Artificial Intelligence. By embedding AI-driven automation, prediction, and optimization into each phase of SDLC, the project enhances productivity, reduces errors, improves code quality, and accelerates delivery timelines. SmartSDLC ensures seamless collaboration across development, testing, and operations teams while providing intelligent recommendations for better decision-making.
* **Features**:
  + **AI-Powered Requirement Analysis**
  + *Key Point*: Intelligent requirement gathering
  + *Functionality*: Uses Natural Language Processing (NLP) to convert client requirements into structured specifications.
  + **Automated Code Generation**
  + *Key Point*: Faster development
  + *Functionality*: Generates boilerplate code and reusable templates based on user stories.

**AI-based Testing Assistant**

* + *Key Point*: Smart test case creation
  + *Functionality*: Creates automated test cases, predicts defects, and suggests bug fixes.

**Project Timeline Predictor**

* + *Key Point*: Delivery forecasting
  + *Functionality*: Uses ML to predict project delivery dates, bottlenecks, and risks.

**Code Quality Analyzer**

* + *Key Point*: Continuous improvement
  + *Functionality*: Identifies vulnerabilities, enforces coding standards, and provides optimization tips.

**Collaboration Dashboard**

* + *Key Point*: Unified team workspace
  + *Functionality*: Provides real-time progress tracking, reporting, and AI-driven recommendations.

**3. Architecture**

* **Frontend (React/Streamlit)**:  
   Provides an intuitive dashboard for project management, requirements tracking, and reporting. Supports drag-and-drop document uploads and AI-driven suggestions.
* **Backend (FastAPI/Node.js)**:  
   Handles business logic, API endpoints, code analysis, testing automation, and AI model integration.
* **AI/ML Models**:
  + Requirement analysis using NLP
  + Project forecasting with regression models
  + Defect prediction using classification algorithms
  + Code quality scoring with deep learning
* **Database (PostgreSQL/MongoDB)**:  
   Stores requirements, project timelines, test results, and code analysis data.
* **Version Control & CI/CD (GitHub, Jenkins)**:  
   Ensures continuous integration, testing, and automated deployment.

**4. Setup Instructions**

**Prerequisites**:

* Python 3.9 or Node.js 16+
* FastAPI / Express.js
* React / Streamlit
* Docker & Kubernetes (for deployment)
* Git & CI/CD tools

**Installation Process:**

* Clone the repository
* Install dependencies from requirements.txt / package.json
* Configure .env file with API keys and database credentials
* Run backend server with FastAPI/Node.js
* Launch frontend dashboard (React/Streamlit)
* Start interacting with AI-powered modules

**5. Folder Structure**

app/ – Backend logic (API routes, models, services) ui/ – Frontend components and dashboard ai\_models/ – ML models for prediction and analysis docs/ – Project documentation and architecture diagrams tests/ – Automated test scripts deploy/ – Docker and CI/CD configurations

**6. Running the Application**

* Start the backend with FastAPI/Node.js
* Run the frontend dashboard
* Access project management tools through sidebar navigation
* Upload requirement documents for AI analysis
* View generated code snippets, test cases, and reports
* Monitor project progress with AI-driven predictions

**7. API Documentation**

* **POST /analyze-requirements** – Converts client inputs into structured requirements
* **POST /generate-code** – Produces boilerplate code from specifications
* **POST /predict-delivery** – Provides project timeline predictions
* **GET /analyze-code** – Returns quality, vulnerabilities, and fixes
* **POST /run-tests** – Executes AI-generated test cases

**8. Authentication**

* JWT Token-based authentication
* Role-based access (Admin, Developer, Tester, Manager)
* OAuth2 for secure API access
* Planned enhancements: biometric/SSO integration

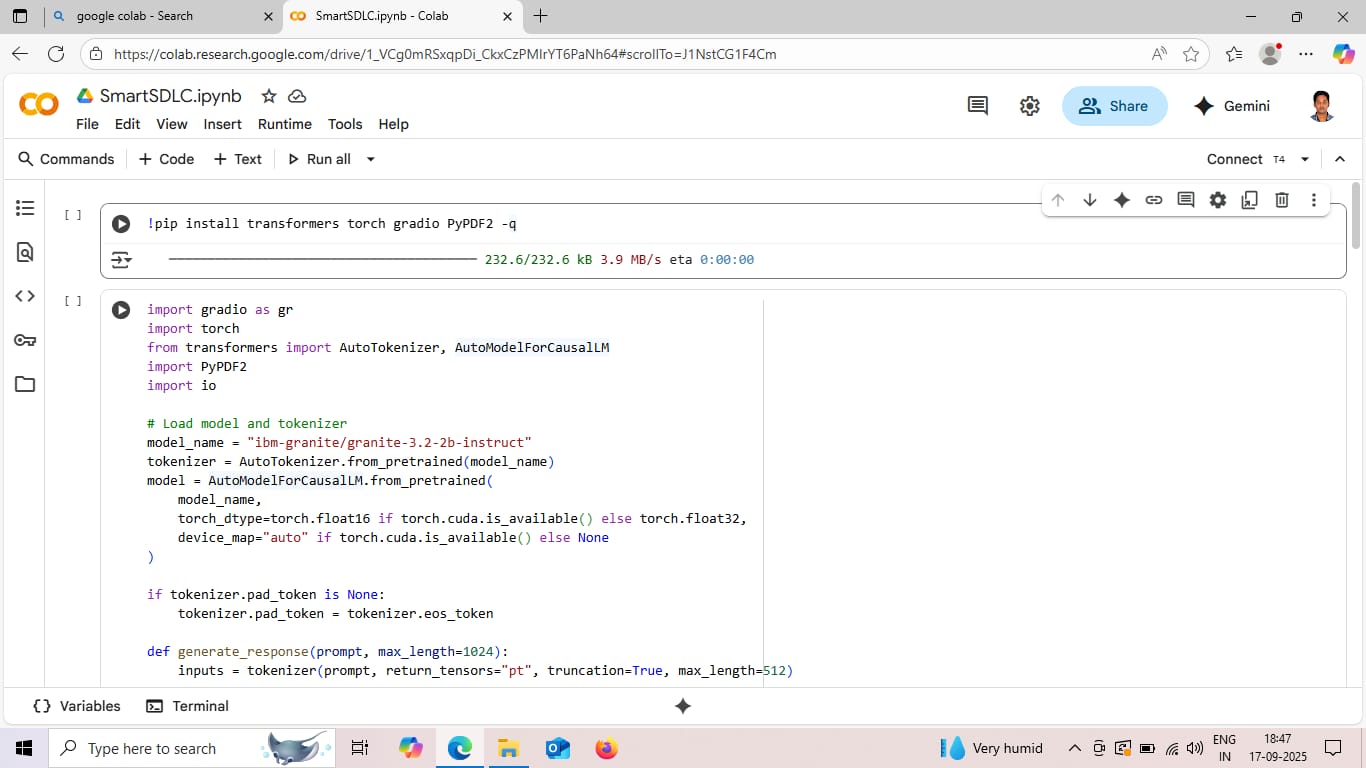
**9. User Interface**

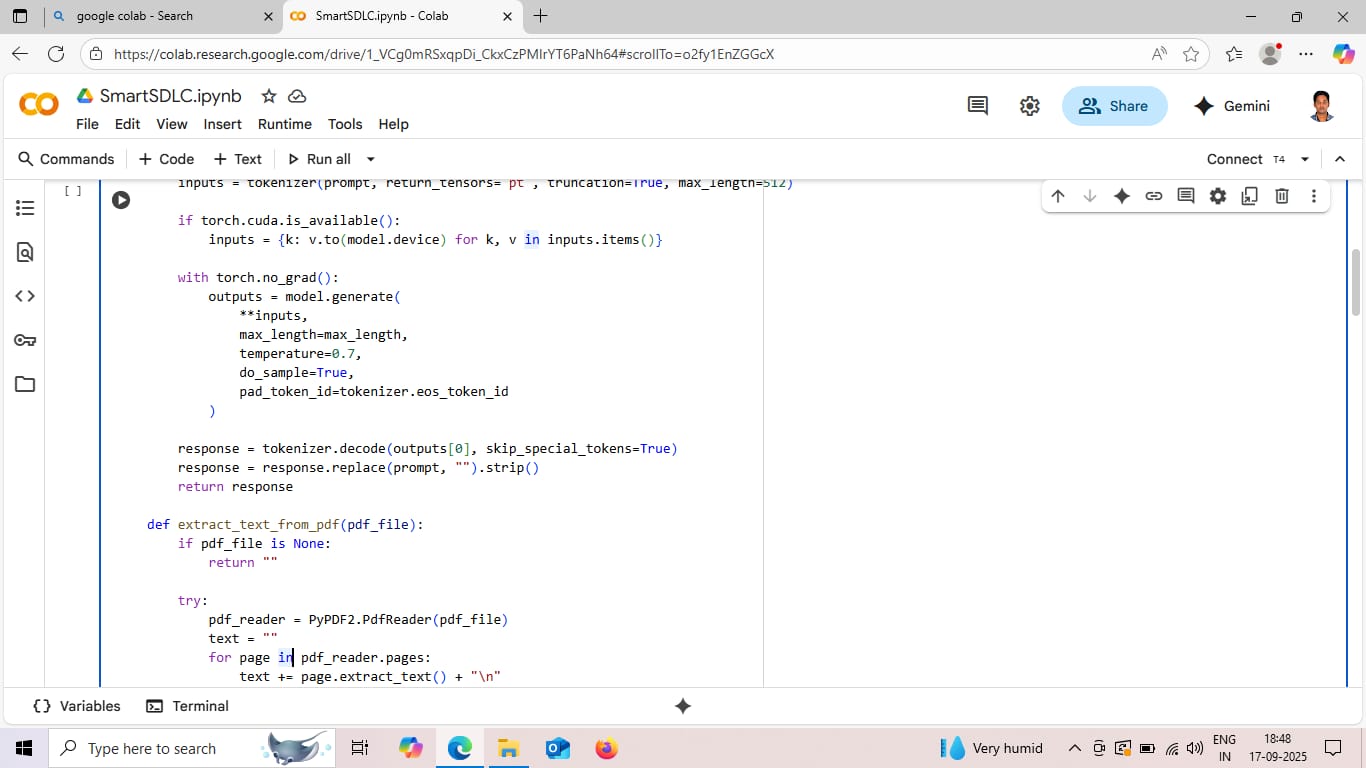
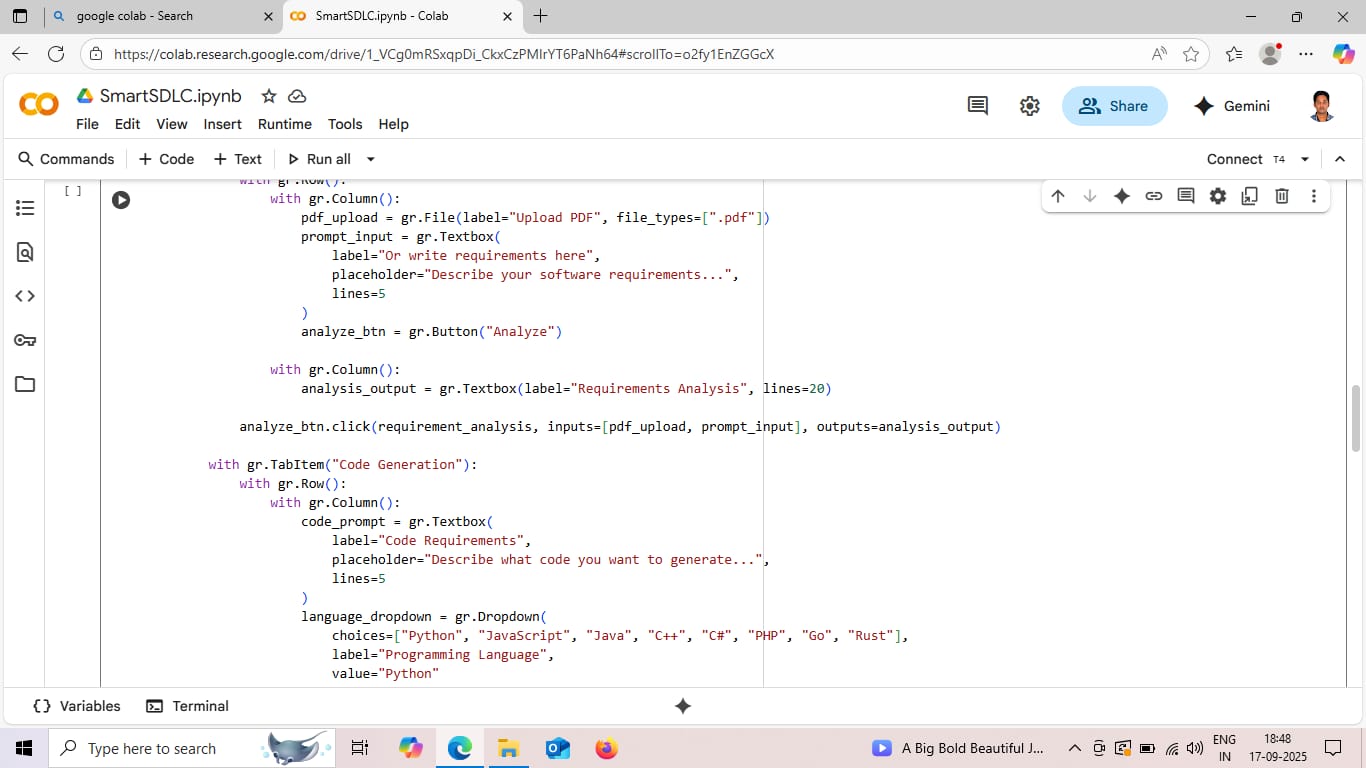
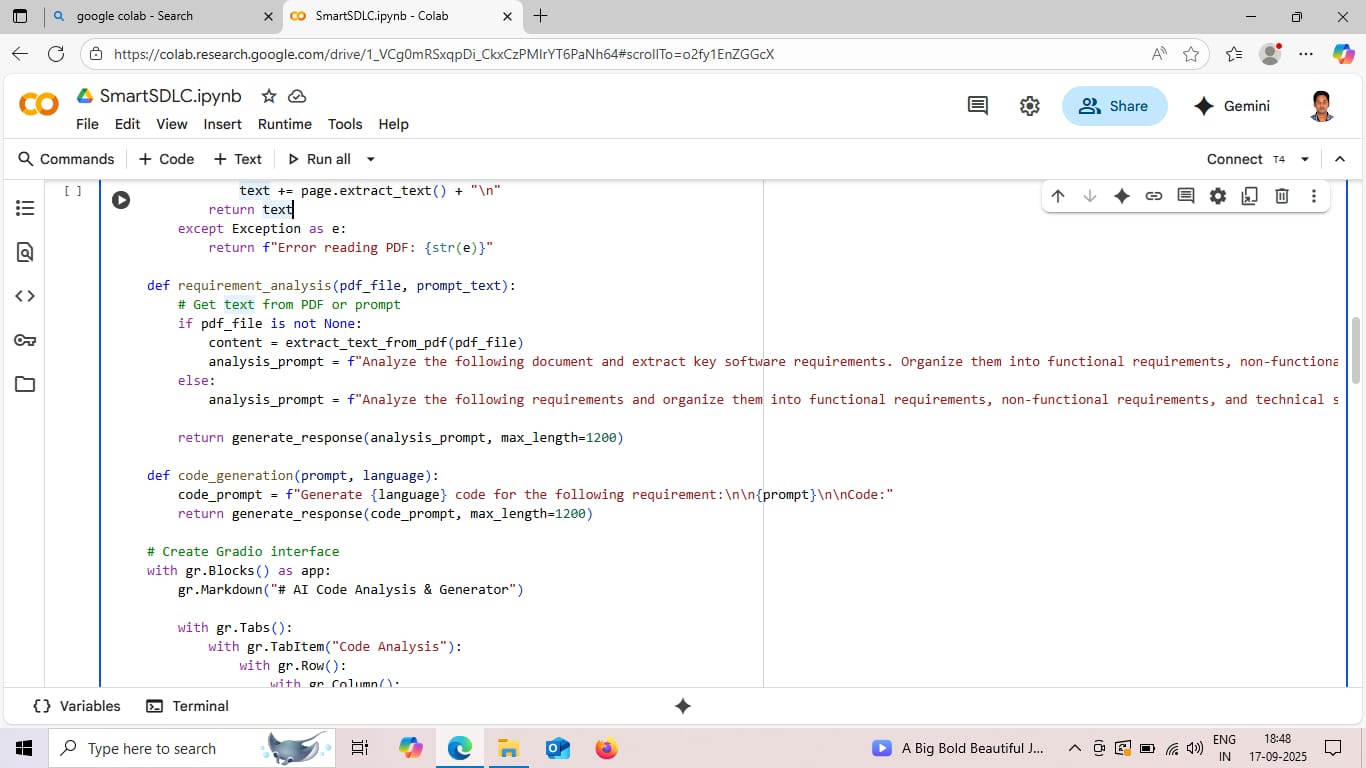
* Sidebar navigation with modules: Requirements, Code, Testing, Reports
* Real-time dashboards with charts & AI insights
* Automated bug-tracking and alerts
* Downloadable project health reports

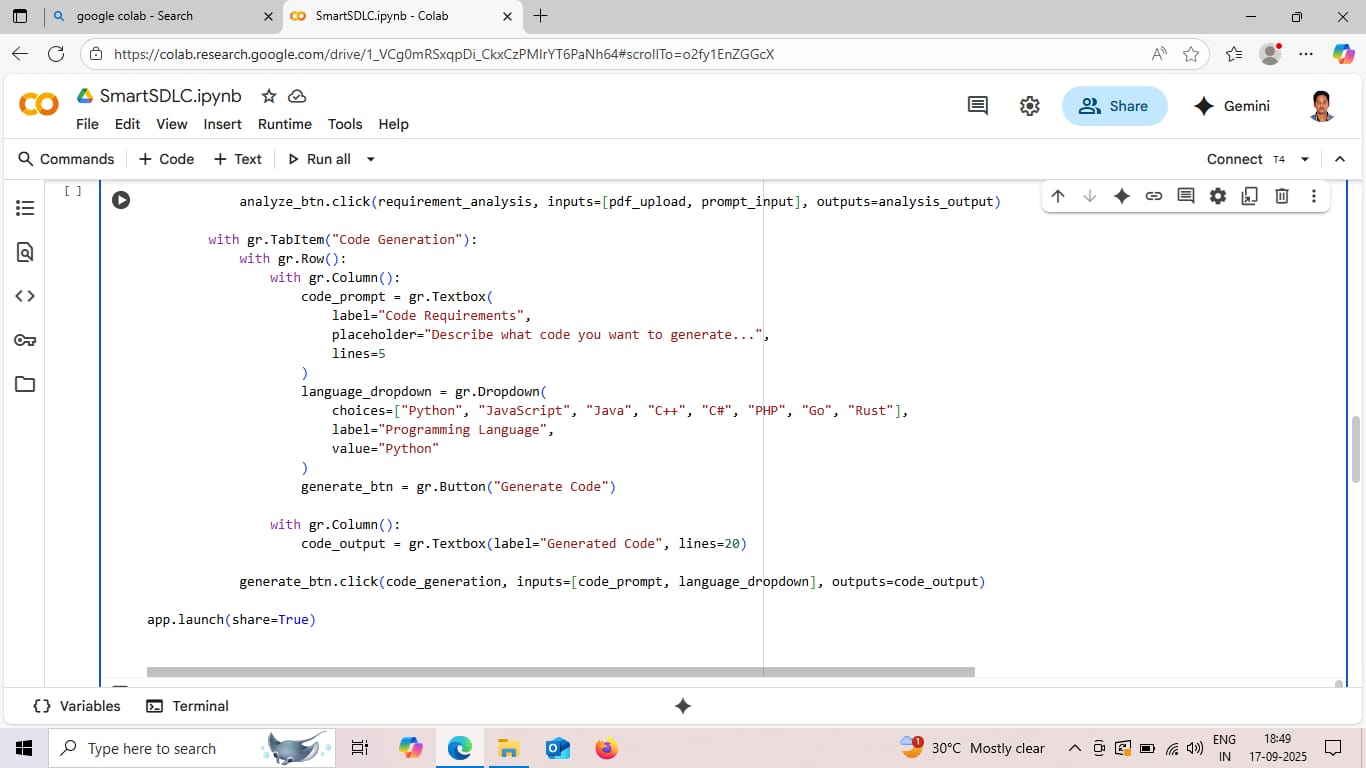
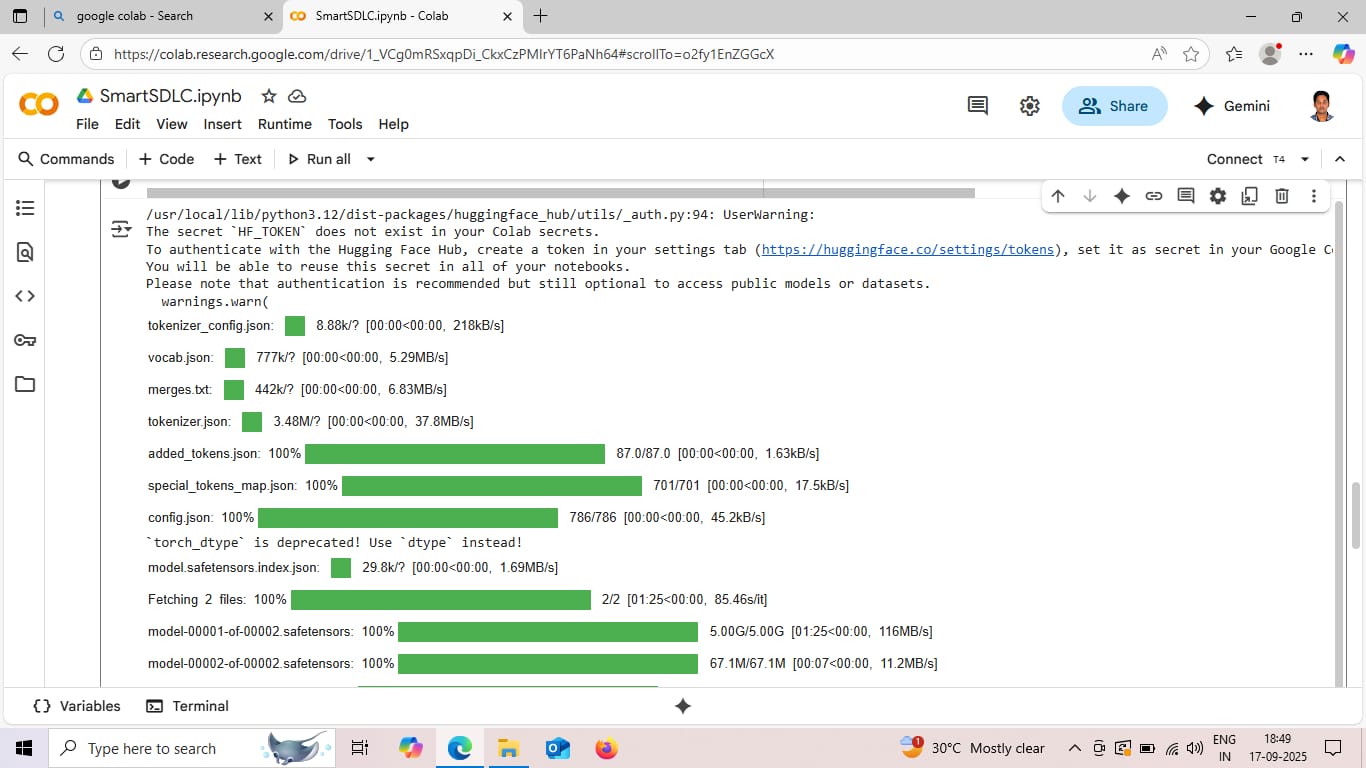
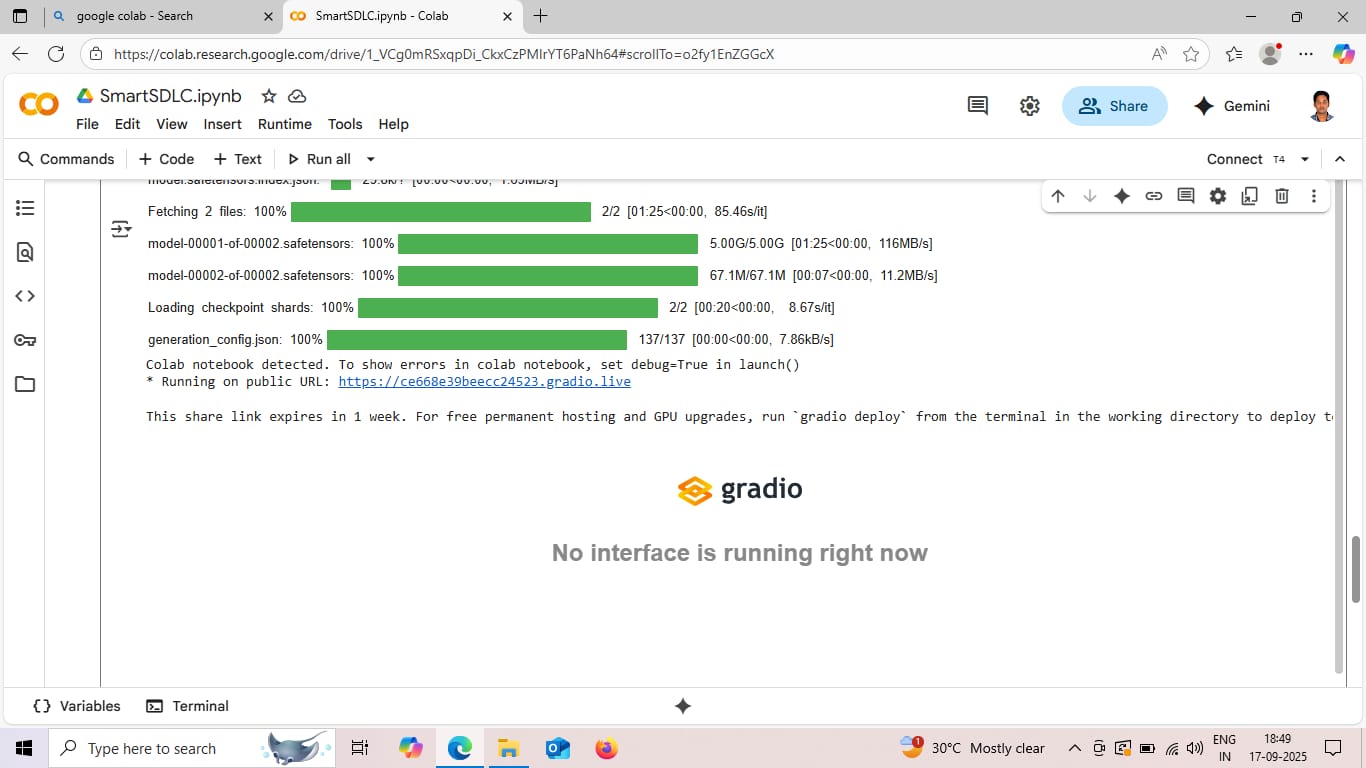
**10. Testing**

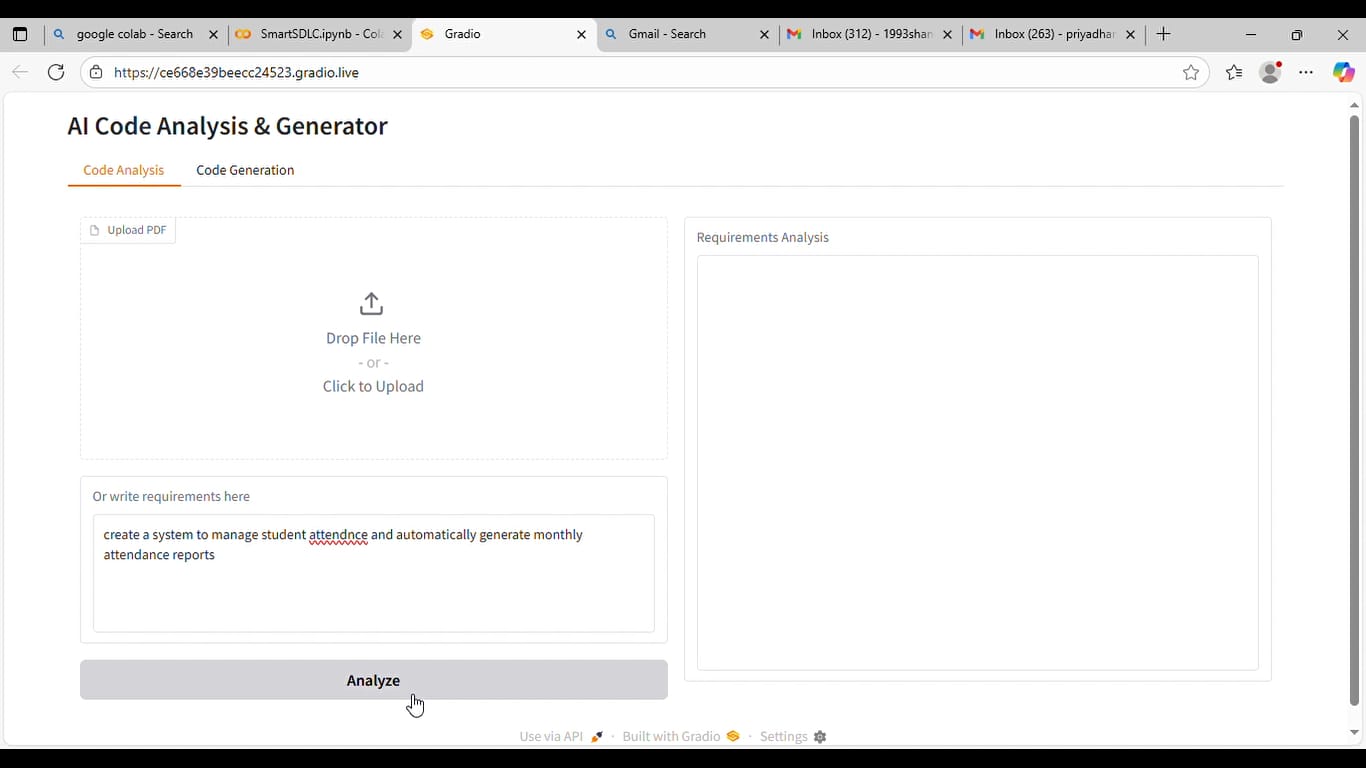
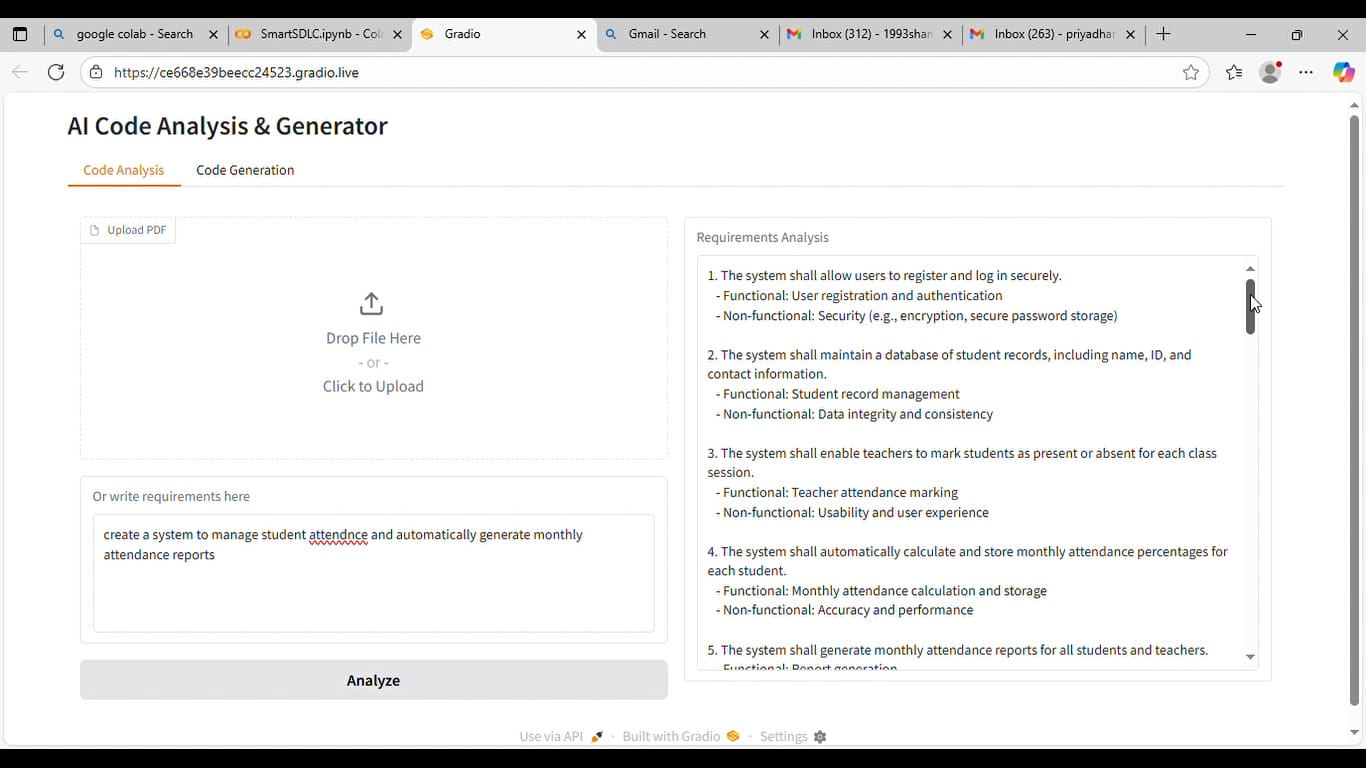
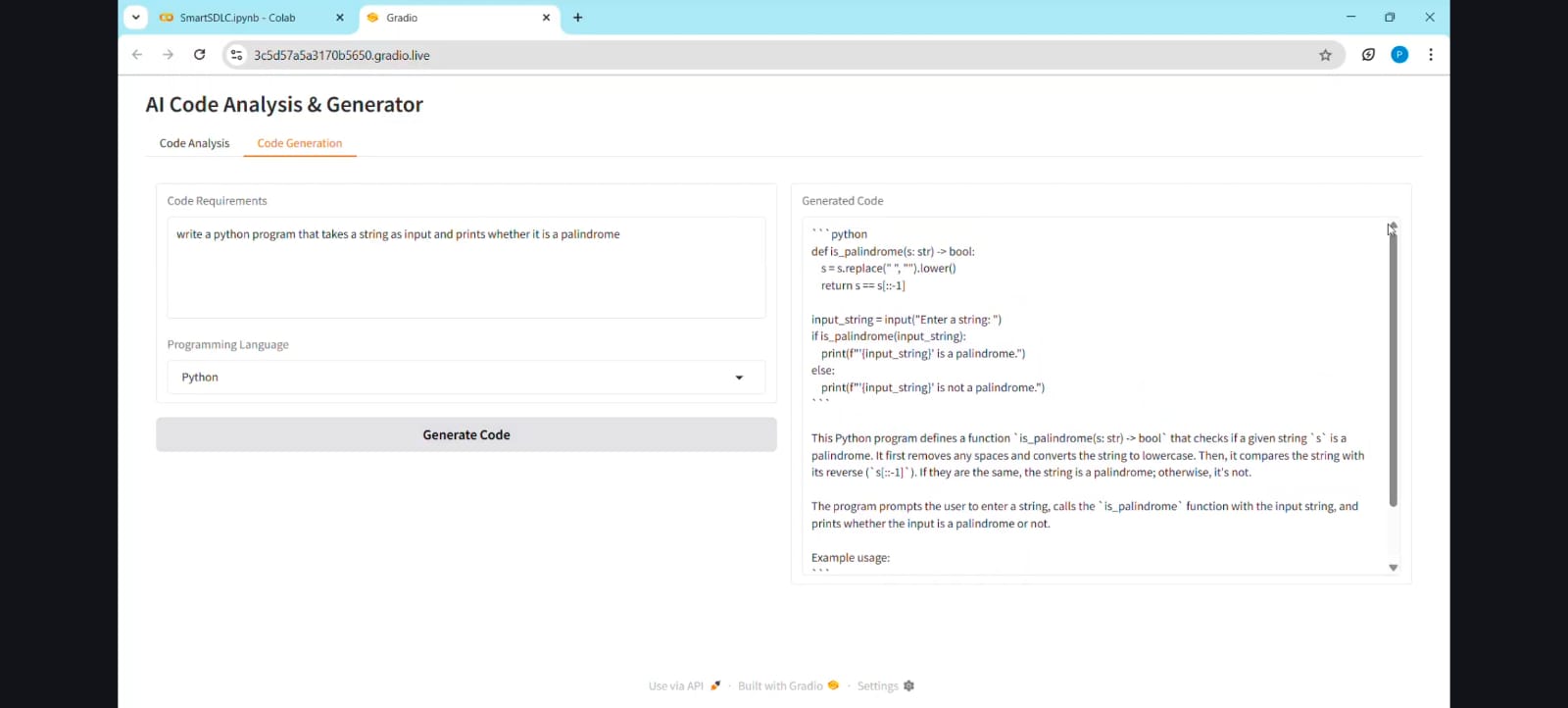
* **Unit Testing** – For requirement parser, code generator, and ML models
* **Integration Testing** – For frontend-backend API interactions
* **System Testing** – To ensure SDLC flow works end-to-end
* **Performance Testing** – To validate efficiency of AI models
* **Edge Case Testing** – For incomplete requirements and ambiguous inputs

**11. Screenshots**

****

****

****

****

**12. Known Issues**

* High dependency on quality of input data
* AI-generated code may need manual review
* Initial training of ML models requires significant data

**13. Future Enhancements**

* Voice-enabled requirement gathering
* Integration with cloud-based DevOps tools (AWS, Azure DevOps)
* Self-healing CI/CD pipelines
* AI-based project cost estimation