Smart SDLC Documentation

1. Introduction

• Project Title: Smart Software Development Life Cycle (Smart SDLC)
• Team leader: S.Keerthana
• Team Member: M.vanitha
• Team Member: P. Abinaya
• Team Member: G.Dhanya

2. Project Overview

• Purpose:
The Smart SDLC is designed to optimize software development practices by integrating automation, AI-driven decision-making, and agile frameworks. It ensures better planning, efficient development, continuous testing, and real-time monitoring of software projects. The aim is to minimize risks, reduce costs, and deliver high-quality software faster.

• Features:

Agile Planning Assistant
- Key Point: Intelligent sprint and backlog management
- Functionality: AI-driven suggestions for task prioritization and resource allocation

Automated Code Review
- Key Point: Continuous quality checks
- Functionality: Uses ML models to detect bugs, vulnerabilities, and style issues in realtime

CI/CD Pipeline Automation
- Key Point: Streamlined deployment
- Functionality: Automates build, test, and deployment with rollback support

Smart Testing Framework

- Key Point: AI-powered testing
- Functionality: Generates and executes test cases dynamically based on code changes
  Project Health Dashboard
- Key Point: Real-time monitoring
- Functionality: Provides KPIs like velocity, defect rates, and deployment frequency

3. Architecture

Frontend (React/Angular): Provides an interactive UI for project tracking, reports, and dashboards.
Backend (FastAPI/Django): Handles APIs, business logic, and integrations.
AI/ML Integration: AI models for prediction, anomaly detection, and automation.
DevOps Tools: Jenkins, GitHub Actions, or GitLab CI for automated pipelines.
Database: PostgreSQL or MongoDB for storing project, code, and test data.

4. Setup Instructions

Prerequisites:
o Python 3.9 or later
o Node.js and npm
o Docker &
Kubernetes (optional for
deployment) o API keys
for AI/ML modules

Installation Process: o Clone the
repository o Install backend
dependencies o Install frontend
dependencies o Configure environment
variables in .env
o Run backend
server o Run frontend
application

5. Folder Structure

backend/ – Contains all backend APIs and logic backend/api/ –
Modular API routes for sprints, tasks, and testing frontend/ –

React or Angular UI for dashboards and reports ci_cd/ – CI/CD pipeline configuration files
ai_modules/ – ML models for prediction and anomaly detection
tests/ – Automated testing scripts and frameworks

6. Running the Application

➢ Start backend server
➢ Run frontend dashboard
➢ Navigate through project pages
➢ View sprint plans, test cases, reports, and KPIs
➢ Deploy and monitor CI/CD pipelines in real-time

7. API Documentation

POST /sprint/plan – Generates sprint backlog
POST /code/review – Submits code for automated review
GET /metrics/health – Retrieves project health KPIs
POST /test/execute – Runs AI-generated test cases
POST /deploy – Triggers deployment pipeline

8. Authentication

• JWT-based authentication for API access
• OAuth2 for third-party integrations
• Role-based access control (Admin, Developer, Tester, Manager)

9. User Interface

The UI provides:

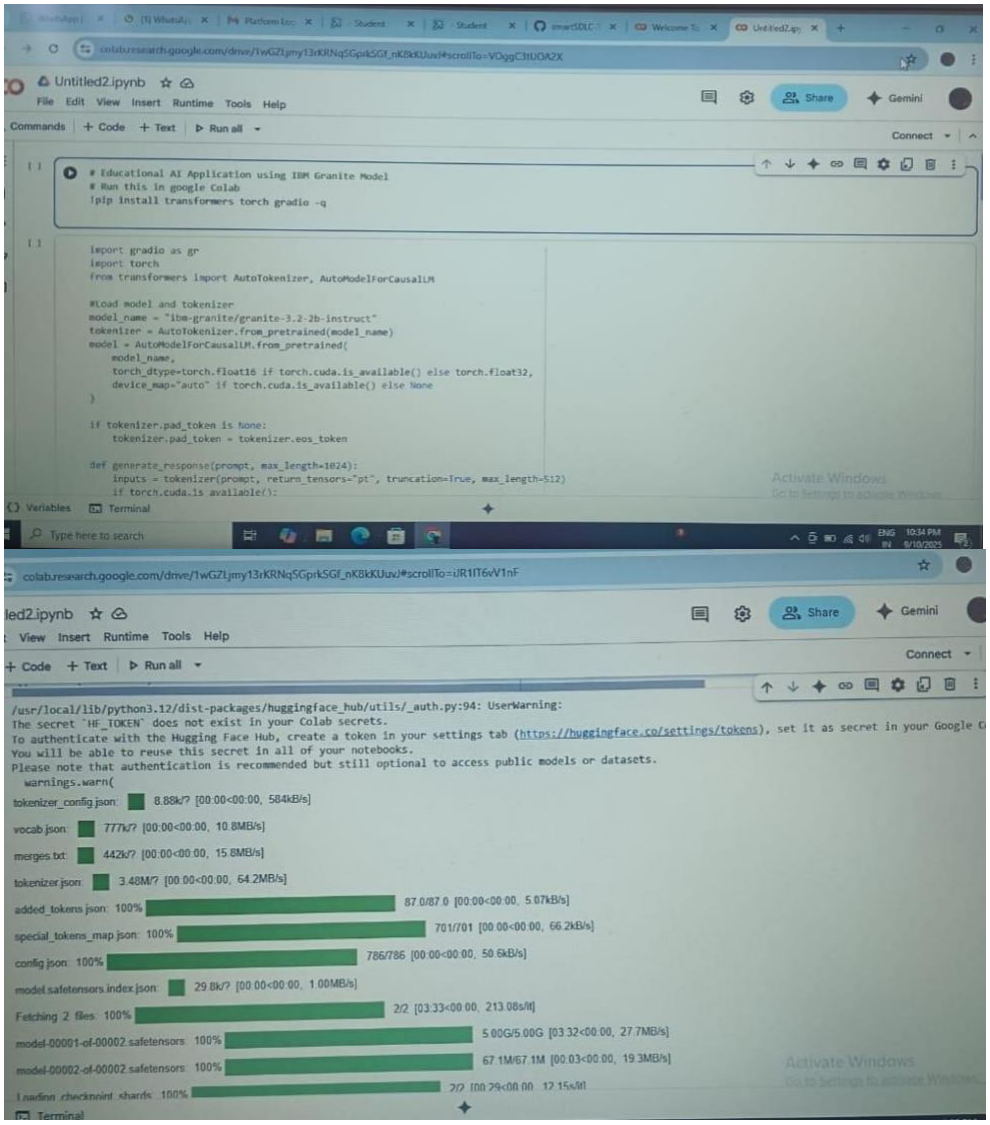
- Dashboard with KPIs and project health
- Tabs for sprint planning, code review, testing, and deployment
- Real-time notifications and alerts - Downloadable reports

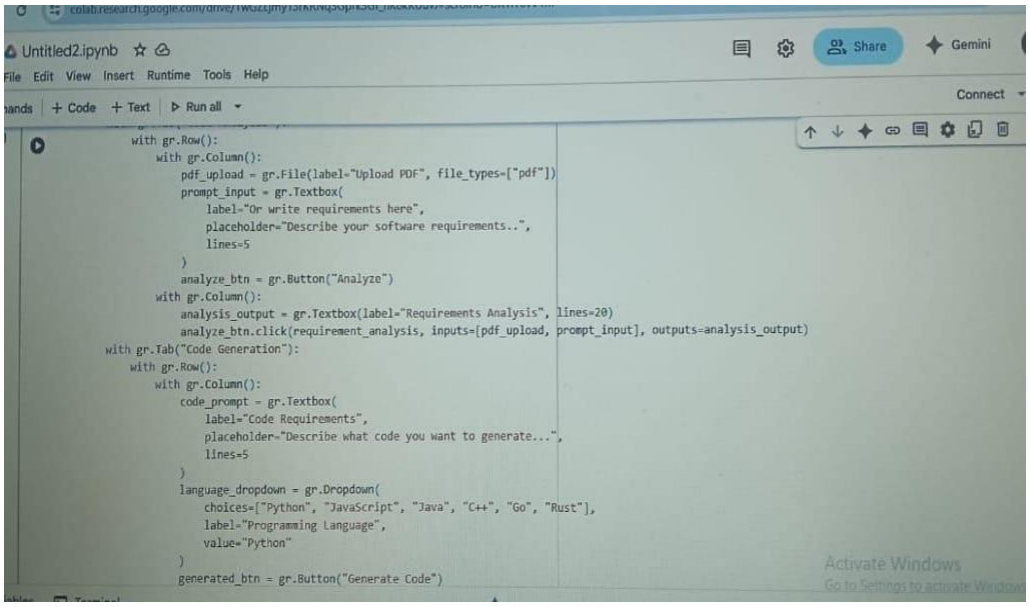
10. Testing

Unit Testing: For utility functions and ML modules
API Testing: Using Postman and automated scripts

Integration Testing: Ensures smooth workflow between modules
Manual Testing: For UI and dashboard validation
Edge Case Handling: Large codebases, failed deployments, malformed inputs

11. Screenshots





12. Known Issues

• Limited support for legacy systems
• High resource usage for AI models
• Some modules require internet access for cloud-based AI service

- 1. *Scope Creep:* Changes in project scope can lead to delays, cost overruns, and decreased quality.

- 2. *Communication Breakdowns:* Poor communication among team members, stakeholders, and customers can lead to misunderstandings and errors.

- 3. *Inadequate Requirements Gathering:* Insufficient or inaccurate requirements can result in a product that doesn't meet user needs.

- 4. *Testing Challenges:* Inadequate testing can lead to defects and bugs in the final product.

- 5. *Delays and Schedule Slippage:* Unforeseen issues, changes, or dependencies can cause project delays.

- 6. *Cost Overruns:* Unexpected expenses, changes, or scope creep can lead to budget overruns.

- 7. *Quality Issues:* Inadequate attention to quality can result in a product that doesn't meet user expectations.

- 8. *Stakeholder Management:* Managing stakeholder expectations, priorities, and communication can be challenging.

- 9. *Technical Debt:* Quick fixes, workarounds, or incomplete solutions can lead to technical debt, making future maintenance and updates more difficult.

- 10. *Change Management:* Managing changes, updates, and iterations can be complex, especially in large projects.

13. Future Enhancements

- Support for multi-cloud DevOps pipelines
- More advanced AI models for predictive analytics

- Integration with additional project management tools (e.g., Jira, Trello)
- Voice-enabled project assistant