Smart SDLC Documentation

• Introduction

• Project Title: Smart Software Development Life Cycle (Smart SDLC)
• Team leader: K.Valarmathi
• Team Member: S.Santhiya
• Team Member: I.Monisha
• Team Member: C.Sasikala

• 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 real- time

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

• 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.

• Setup Instructions

Prerequisites:

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

Installation Process:

- Clone the repository
- Install backend dependencies
- Install frontend dependencies
- Configure environment variables in .env
- Run backend server
- Run frontend application

• 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

• 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

• 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

• Authentication

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

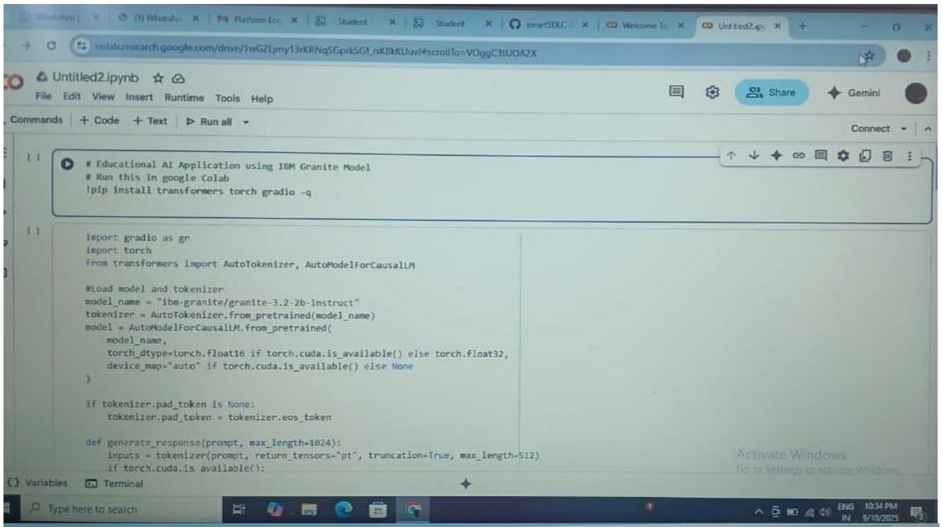
• User Interface

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

• Testing

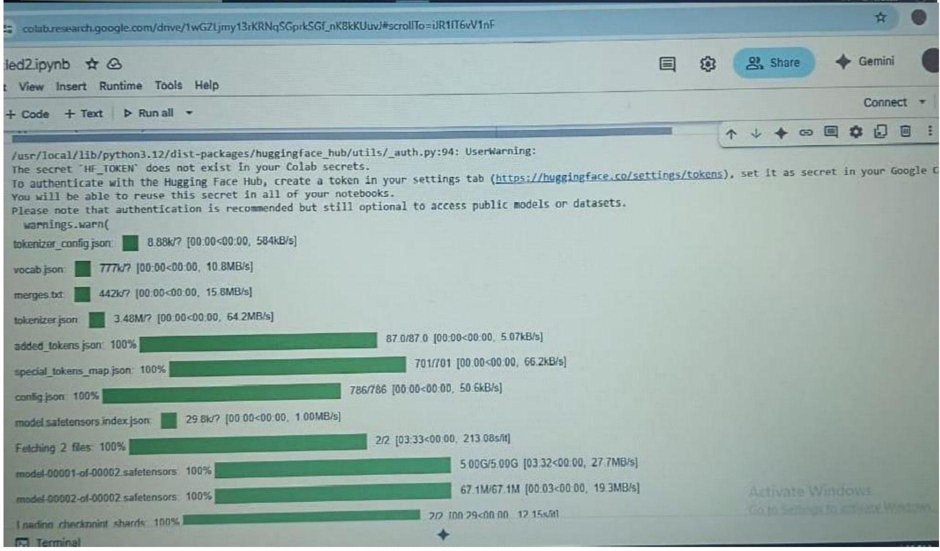
Unit Testing: For utility functions and ML modules API
Testing: Using Postman and automated scripts
Integration Testing: Ensures smooth workflow between modules Manual
Testing: For UI and dashboard validation
Edge Case Handling: Large codebases, failed deployments, malformed inputs



Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
Connect
Variables
Terminal
Type here to search

```
# Educational AI Application using IBM Granite Model
# Run this in google Colab
!pip install transformers torch gradio -q
```

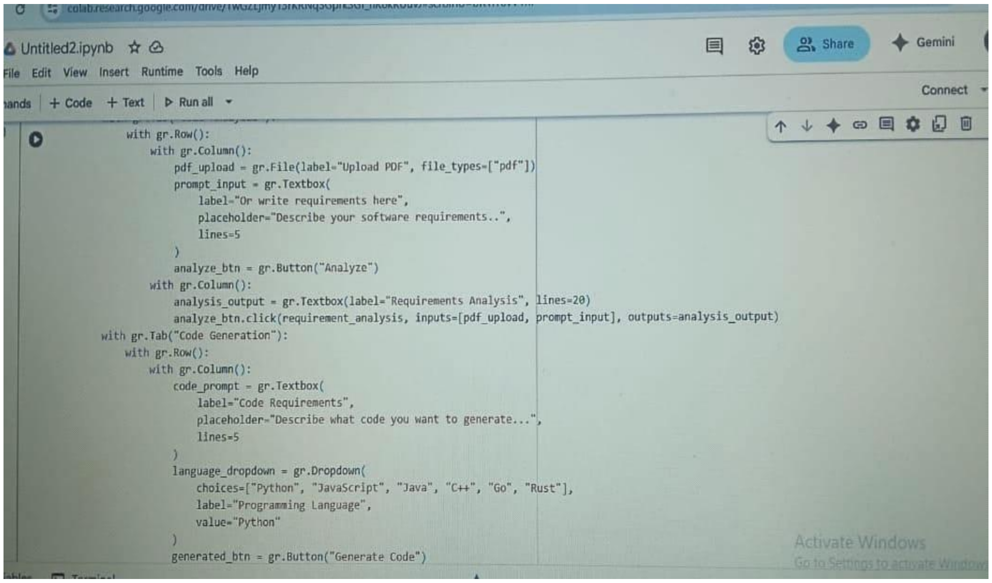
```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

#Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
    if torch.cuda.is_available():
```



colab.research.google.com/drive/1wGZLjmy13rKRNqSGprkSGf_nK8kKUuvJ#scrollTo=iJR1IT6vV1nF
led2.ipynb
View Insert Runtime Tools Help
+ Code + Text ▷ Run all
Share
Gemini
Connect
Terminal

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google C
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json:     8.88k/? [00:00<00:00, 584kB/s]
vocab.json:     777k/? [00:00<00:00, 10.8MB/s]
merges.txt:     442k/? [00:00<00:00, 15.8MB/s]
tokenizer.json:     3.48M/? [00:00<00:00, 64.2MB/s]
added_tokens.json: 100%     87.0/87.0 [00:00<00:00, 5.07kB/s]
special_tokens_map.json: 100%     701/701 [00:00<00:00, 66.2kB/s]
config.json: 100%     786/786 [00:00<00:00, 50.6kB/s]
model.safetensors.index.json:     29.8k/? [00:00<00:00, 1.00MB/s]
Fetching 2 files: 100%     2/2 [03:33<00:00, 213.08s/it]
model-00001-of-00002.safetensors: 100%     5.00G/5.00G [03:32<00:00, 27.7MB/s]
model-00002-of-00002.safetensors: 100%     67.1M/67.1M [00:03<00:00, 19.3MB/s]
Loading checkpoint shards: 100%     2/2 [00:29<00:00, 12.15s/it]
```

- **Screenshots**



Untitled2.ipynb
File Edit View Insert Runtime Tools Help
nands + Code + Text ▷ Run all
Share
Gemini
Connect
Terminal

```
        with gr.Row():
            with gr.Column():
                pdf_upload = gr.File(label="Upload PDF", file_types=["pdf"])
                prompt_input = gr.Textbox(
                    label="Or write requirements here",
                    placeholder="Describe your software requirements..",
                    lines=5
                )
                analyze_btn = gr.Button("Analyze")
            with gr.Column():
                analysis_output = gr.Textbox(label="Requirements Analysis", lines=20)
                analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input], outputs=analysis_output)
    with gr.Tab("Code Generation"):
        with gr.Row():
            with gr.Column():
                code_prompt = gr.Textbox(
                    label="Code Requirements",
                    placeholder="Describe what code you want to generate...",
                    lines=5
                )
                language_dropdown = gr.Dropdown(
                    choices=["Python", "JavaScript", "Java", "C++", "Go", "Rust"],
                    label="Programming Language",
                    value="Python"
                )
                generated_btn = gr.Button("Generate Code")
```

Known Issues

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

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