ProjectTitle:SmartSDLC–AI-Enhanced Software Development Lifecycle.

Introduction:

Team ID : NM2025TMID06797

Team Size : 4

Team Leader : DHIVYADHARSHINI M

Team member : INDUJA C

Team member : GOWRI S

Team member : ABINAYA T

Sure!Here'sacomprehensiveexplanationofthe **SmartSDLC–AI-EnhancedSoftware Development Lifecycle** project, covering the following:

1. ProjectOverview
2. Architecture
3. SetupInstructions
4. FolderStructure
5. RunningtheApplication
6. APIDocumentation
7. Authentication
8. UserInterface
9. Testing

✅1.Project Overview

SmartSDLC is a full-stack software development platform powered by **Artificial Intelligence**thatassistsdevelopers,testers,andprojectmanagersthroughoutthe **Software Development Lifecycle (SDLC)**.

ItleveragesAIto:

- Extract&validaterequirementsfromnaturallanguage input
- Generatesystemdesigndiagramsandboilerplatecode
- Auto-generatetestcases

- Predictbugsanddeploymentissues
- AutomateCI/CDpipelinesand monitoring

□2.Architecture

□SmartSDLCModularArchitecture

```
[FrontendUI-React/Next.js]
         |
         v
[BackendAPI-Node.js/Python(Flask/FastAPI)]
         |
         +-----------------------------+
         |                             |
[AI Engine -NLP, ML Models,LLMs]       [Database- MongoDB /PostgreSQL]
         |
         v
[AIModulesforEachSDLCPhase]
  ├─RequirementAnalyzer (NLP)
  ├─DesignGenerator(LLM+DiagrammingTool)
  ├─CodeGenerator(LLM/GPT+TemplateEngine)
  ├─ TestGenerator (AITest Engine)
  ├─DeploymentOptimizer(DevOpsAI)
  └─Monitoring&Feedback(AIOps)
```

Milestone1:ModelSelectionandArchitecture

Milestone 1 focuses on selecting the optimal AI models for SmartSDLC, choosing granite-13b-chat-v1 for natural language tasks and granite-20b-code-instruct for code generation. The architecture integrates the backend (FastAPI), frontend (Streamlit), AI layer (Watsonx), and modules (LangChain, GitHub). The development environment is set up with necessary dependencies,APIkeys,andamodularstructure,enablingefficientintegrationanddevelopment in subsequent phases.

Activity1.1:Researchandselecttheappropriate generativeAImodel

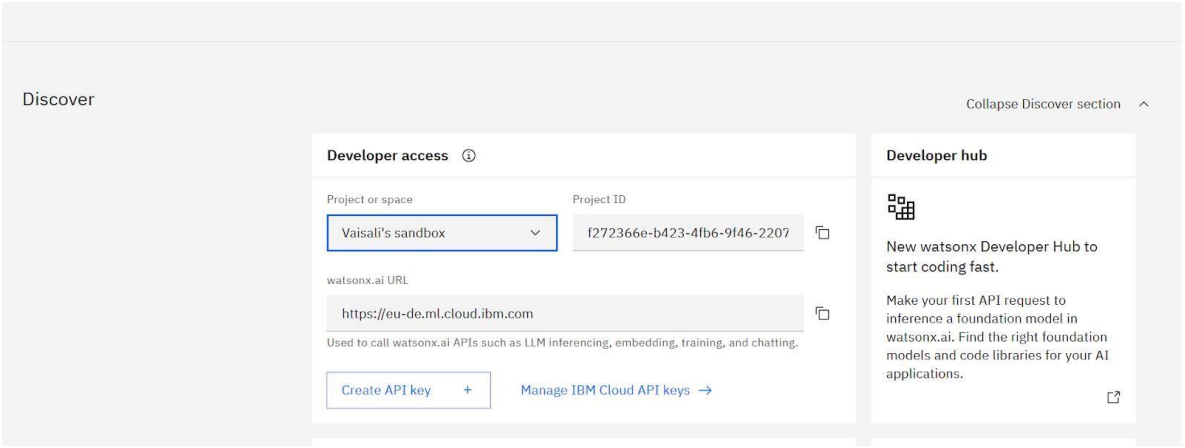
1. **Understandthe ProjectRequirements:**
   Analyzethegoals oftheSmartSDLC platform, particularly theneed for automating SDLC phases likerequirement analysis, codegeneration, test creation,bugfixing,anddocumentation.Thisunderstandinghelpsnarrow down which generative AI models best suit each task.

2. **ExploreAvailableModels:**
   Study IBM's Watsonx Granite model documentation to compare available foundation models. Focus on those capable of handling natural language processingandcodesynthesis,suchasgranite-13b-chat-v1forlanguage understanding and granite-20b-code-instruct for structured code generation andsummarization.

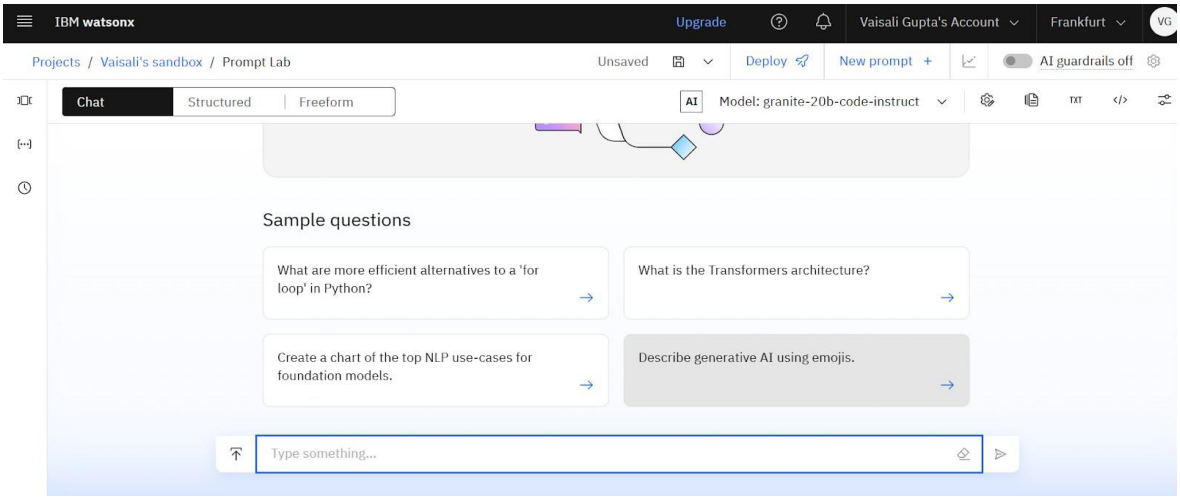
3. **EvaluateModelPerformance:**
   Evaluatemodels based onmetrics likeaccuracy, latency, and quality of generatedresponsesacrossdifferentSDLCtasks.Usepriorbenchmarksorrun pilotqueries(e.g.,"generateafunctiontovalidateemailinput")toassess modelsuitability.

4. **SelectOptimalModels:**
   Finalizethetwobest-fitGranitemodels—granite-13b-chat-v1forinteracting with natural language prompts and granite-20b-code-instruct for multilingual and functional code outputs. These models will power the core logic of SmartSDLC's automation system.

5. **Generating WatsonxaiAPIkey**



6. **Selectingthemodel**



Activity 1.2: Define the architecture of the application.

1. **DraftaSystemArchitectureDiagram:**

   Create a high-level visual representation that includes the backend (FastAPI), the frontend (Streamlit), the AI layer (Watsonx APIs), and service modules (LangChain, GitHubintegration).Includeflowdirectionbetweencomponentssuchasfileuploads,AI interactions, and response rendering.

2. **DefineFrontend-BackendInteraction:**

DetailhowtheStreamlitUIcapturesuserinputs(PDFuploads,textprompts, buggy code) and sends them to FastAPI endpoints. Also, document how the backendrouteshandlelogicandreturnstructuredoutputslikeuserstoriesor codeblocks.

3. **BackendServiceResponsibilities:**
   Definerolesofeachbackendmodule:

   a. PDFprocessingandclassificationlogic(usingPyMuPDF).
   b. PromptgenerationandAPIcommunicationwithWatsonx.
   c. LangChainagentfororchestration.
   d. Authmoduleforusersessionsandhashedlogins.
4. **AIIntegrationFlow:**
   OutlinehowAPI calls aremadetoWatsonx(oroptionally LangChain),and howresponsesareparsedanddisplayedbacktotheuser.Specifyiffallback models or retries will be incorporated.

Activity1.3:Setup the development environment:

1. **InstallPythonandPip:**
   EnsurePython3.10isinstalledforcompatibilitywithWatsonxSDK.Installpip to manage dependencies like FastAPI, Streamlit, and PyMuPDF.
2. **CreateaVirtualEnvironment:**
   Usevenvtocreateisolatedenvironmentsforbackendandfrontend:

   python -mvenvmyenv
   myenv\Scripts\activate

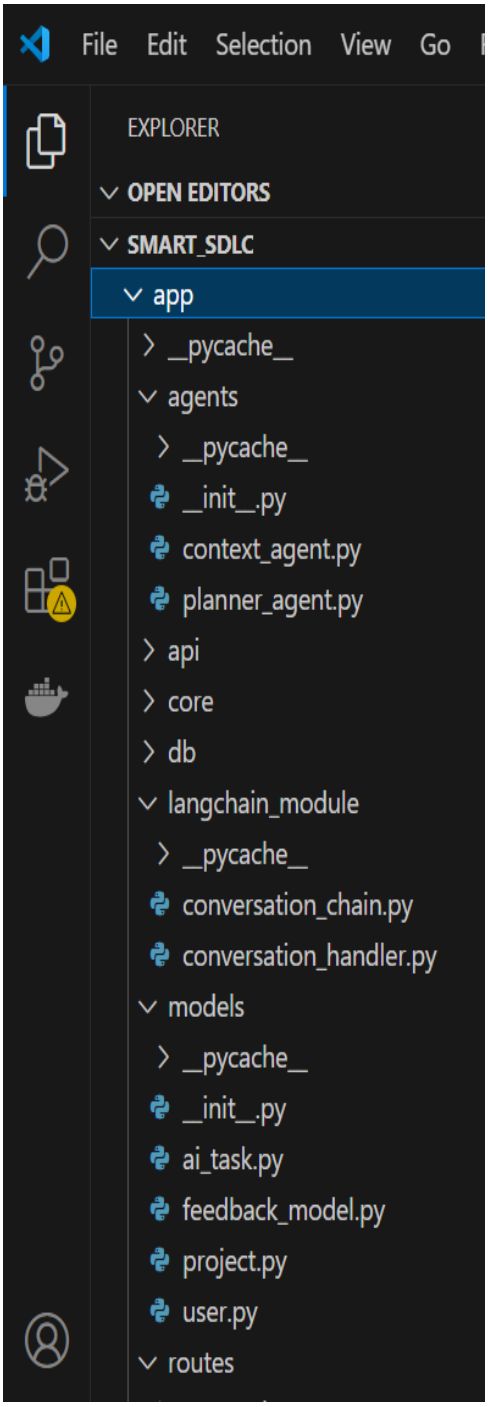
3. **InstallRequiredLibraries:**
   Installallrequiredpackagessuchasfastapi,uvicorn,pymupdf,requests, langchain, streamlit, and ibm-watsonx-aiusing pip. Storethesein requirements.txt.

4. **SetUpAPIKeysandEnvFiles:**
   GenerateIBMCloudAPIkeyandstoreitsecurelyina.envfile.Includeother

configslikemodelIDsandLangChainagentsettings.

5. **OrganizeProjectStructure:**
   Setuptheinitialfolderstructureforbothfrontend(smart_sdlc_frontend/)and backend(app/),ensuringproperseparationofapi/,services/,models/,and utils/ modules for clean development.

filestructure



Milestone2:CoreFunctionalitiesDevelopment

Milestone2involvesbuildingthecoreAI-drivenfeaturesofSmartSDLC,includingrequirement analysis, code generation, test case creation, bug fixing, code summarization, chatbot assistance, andGitHubintegration. Thesefunctionalities areimplementedusingWatsonx and LangChain via modular service scripts. The FastAPI backend handles routing, authentication, and smooth API interactions, connecting frontend inputs with AI processing and generating structured outputs.

Activity2.1:Developthecorefunctionalities:

This activity focuses on building the logic behind each AI-powered SDLC feature. These functionalities aretheheartofSmartSDLCandareimplementedusingIBMWatsonxand LangChain integration.

1. AI-PoweredRequirementAnalysis

- **Module:**ai_story_generator.py
- **Description:**ExtractstextfromuploadedPDFsandusesAItoclassifyeach sentenceintoSDLCphases(Requirements,Design,Development,etc.).Thenit convertsrelevantsentencesintostructureduserstoriestobeusedinplanning or code generation.

```python
@router.post("/upload-pdf")
async def upload_pdf(file: UploadFile = File(...)):
    contents = await file.read()

    with open("temp.pdf", "wb") as f:
        f.write(contents)

    doc = fitz.open("temp.pdf")
    full_text = "".join([page.get_text() for page in doc])
    doc.close()
```

app > services > ai_story_generator.py

```python
# services/ai_story_generator.py

def generate_user_story(requirement: str) -> str:
    # Dummy version — you can replace with Watsonx/OpenAI later
    return f"As a user, I want to {requirement.lower()} so that I can achieve my goal."

```

2. MultilingualCodeGeneration

- **Module:**code_generator.py

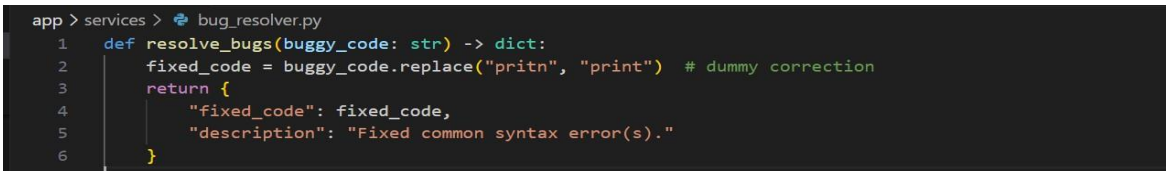
- **Description:**Takestaskdescriptionsoruserstoriesandgeneratesclean, production-ready codeinPython orotherlanguages usinggranite-20b-code-instruct. Idealfor initialdevelopment tasks or boilerplategeneration.

3. AI-DrivenTestCase Generation

- **Module:**Likelypartofcode_generator.py
- **Description:**Generatesunittestsorintegrationtestcasesbasedonthe generatedoruser-providedcode.Helpsdeveloperquicklyvalidatecorelogic.

4. BugFixingandCodeCorrection

- **Module:**bug_resolver.py
- **Description:**Takesbuggycodesnippetsandautomaticallyidentifiesandfixes errors usingAI.Returns a cleanversionofthecodealongwith optional explanations.



5. CodeSummarizationandDocumentation

- **Module:**doc_generator.py
- **Description:** Extracts summaries or generates documentation-style descriptionsofcodeblocks.Helpswithcodereadabilityandmaintainability.

6. Chatbot Assistance

- **Modules:**conversation_handler.py,chat_routes.py
- **Description:**AfloatingAIchatbothelpsusersnavigatetheplatform,answer questions, or give suggestions on SDLC tasks. Powered by LangChain and Watsonxintegration.

app > langchain_module > conversation_handler.py

```python
from app.langchain_module.conversation_chain import chat_with_granite

# Expanded SDLC-related keywords (case-insensitive)
SDLC_KEYWORDS = [
    "requirement", "user story", "user stories", "design", "architecture",
    "development", "coding", "implementation", "programming", "code",
    "testing", "test case", "test cases", "unit test", "integration test",
    "bug", "debug", "fix", "error", "quality assurance", "qa",
    "deployment", "release", "launch", "production",
    "maintenance", "support", "upgrade", "hotfix", "patch",
    "documentation", "version control", "repository", "git", "jira", "devops",
    "sdlc", "software development life cycle", "software process",
    "agile", "scrum", "waterfall", "kanban", "model", "spiral","SDLC", "v-model"
]

# Check if message is SDLC-related
def is_sdlc_related(message: str) -> bool:
    return any(keyword in message.lower() for keyword in SDLC_KEYWORDS)

# Handle conversation with detailed SDLC guidance
def handle_conversation(message: str) -> str:
    try:
        if not is_sdlc_related(message):
            return (
                "🤖 I specialize in Software Development Life Cycle (SDLC) topics.\n\n"
                "You can ask me about:\n"
                "- 📌 SDLC Phases (Requirements, Design, Development, Testing, Deployment, Maintenance)\n"
                "- 🔄 Methodologies (Agile, Scrum, Waterfall, V-Model, Spiral, Kanban)\n"
                "- 🛠️ Tools (Git, JIRA, Jenkins, Selenium)\n"
```

app > langchain_module > conversation_chain.py

```python
import os
import requests
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# Fetch credentials from .env
API_KEY = os.getenv("WATSONX_API_KEY")
PROJECT_ID = os.getenv("WATSONX_PROJECT_ID")

# Get IAM Token from WatsonX
def get_iam_token(api_key: str) -> str:
    url = "https://iam.cloud.ibm.com/identity/token"
    headers = {"Content-Type": "application/x-www-form-urlencoded"}
    data = {
        "apikey": api_key,
        "grant_type": "urn:ibm:params:oauth:grant-type:apikey"
    }

    try:
        response = requests.post(url, headers=headers, data=data)
        response.raise_for_status()
        token = response.json()["access_token"]
        print("🔐 Token fetched successfully.")
        return token
    except Exception as e:
        print(f"[❌ ERROR fetching token] {e}")
```

7. ProjectFeedbackandLearningLoop

- **Module:**feedback_routes.py,feedback_model.py

- **Description:**CollectsuserfeedbackonAIoutputandstoresitinadatabaseor JSON for improving prompts,models, or usability infutureiterations.

app > models > feedback_model.py

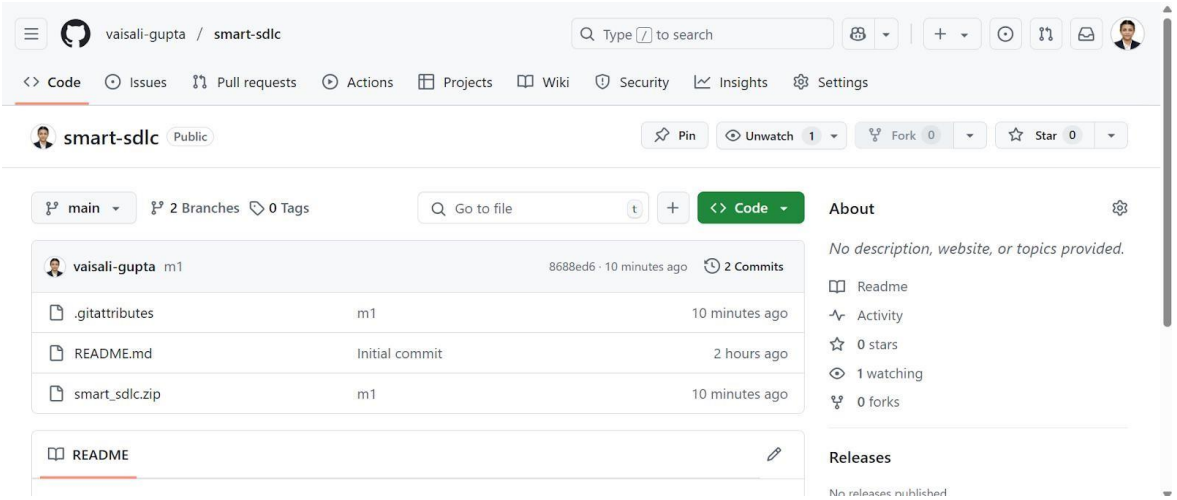
```python
from pydantic import BaseModel

class Feedback(BaseModel):
    name: str = "Anonymous"
    message: str
```

app > routes > feedback_routes.py

```python
from fastapi import APIRouter, HTTPException
from pydantic import BaseModel
import json
import os

router = APIRouter()

# ✅ Path to your feedback file inside the db folder
FEEDBACK_FILE = "app/db/feedback_data.json"

# Create the file if it doesn't exist
if not os.path.exists(FEEDBACK_FILE):
    with open(FEEDBACK_FILE, "w") as f:
        json.dump([], f)

class Feedback(BaseModel):
    username: str
    message: str
    rating: int  # e.g., 1 to 5 stars

@router.post("/feedback")
def collect_feedback(feedback: Feedback):
    try:
        # Load existing feedback
        with open(FEEDBACK_FILE, "r") as f:
            data = json.load(f)
```

8. GitHub Integration

- **Module:**github_service.py
- **Description:**AutomatesGitHubworkflowssuchaspushingAI-generatedcode to a repo, opening issues, or syncing generated documentation.



Output:FastAPIBackend

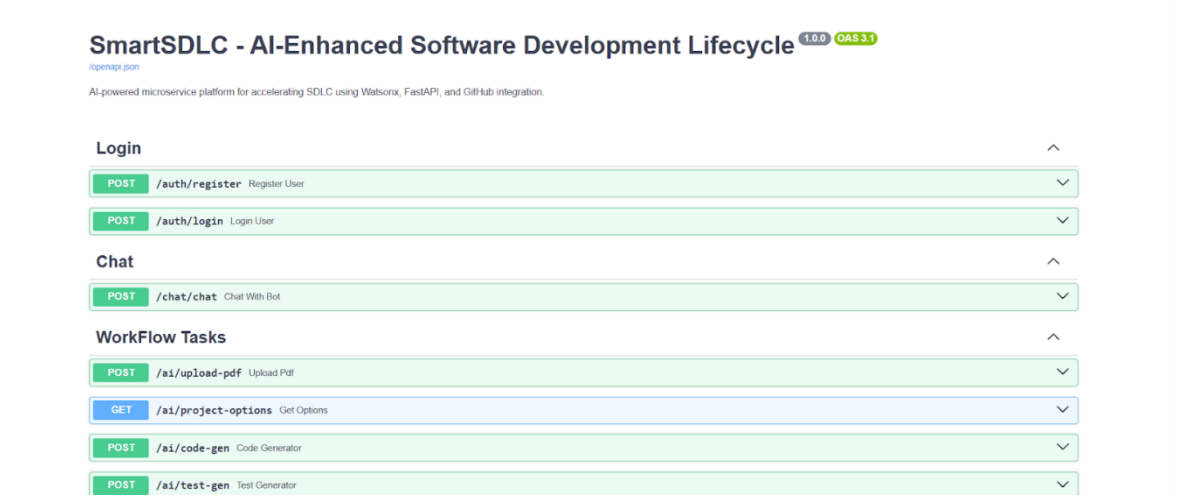


Figure2:SmartSDLCAPIDocumentation–ExploreandinteractwithAI-poweredmicroservices for login, chat, and workflow automation via FastAPI Swagger UI.

Activity2.2:ImplementtheFastAPIbackendto manageroutinganduserinputprocessing, ensuringsmoothAPIinteractions.

Thisactivityfocusesonthe**APIlayerandroutinglogic**thatconnectsuserrequests with the backend services and AI models.

1. BackendRouting

- **Modules:** routes/ai_routes.py, routes/auth_routes.py,routes/chat_routes.py, routes/feedback_routes.py
- **Description:**FastAPIroutershandleincomingPOST/GETrequestsfromthe frontend.Eachendpointcorrespondstoonefeaturesuchas/generate-code, /upload-pdf,/fix-bugs,or/register.

```python
# === AI Services ===
@router.post("/code-gen")
def code_generator(requirements: str = Form(...), language: str = Form("python")):
    return generate_code(requirements, language)

@router.post("/test-gen")
def test_generator(code: str = Form(...), language: str = Form("python")):
    return generate_test(code, language)

@router.post("/fix-bugs")
def bug_fixer(buggy_code: str = Form(...)):
    return fix_bugs(buggy_code)

@router.post("/summarize")
def summarize_code(code: str = Form(...)):
    return summarize_documentation(code)
```

2. UserAuthentication

- **Modules:**auth_routes.py,security.py,user.py
- **Description:** Manages secure user login and registration with hashed passwords.Eachrequestchecksthe user'ssessionorcredentialsbefore executing the corresponding service.

3. ServiceLayerLogic

- **Modules:**services/
- **Description:**Theservicemodulesencapsulatethecorebusinesslogicfor AI processing.Eachroutecallstheappropriateservicefunctionwithcleaneduser input and returns formatted responses.

4. AI&LangChain Integration

- **Modules:**watsonx_service.py,chat_routes.py
- **Description:**Thesemoduleshandleinteractionwithexternal AIservices. Inputsfromroutesareformattedintoprompts,passedtotheWatsonxor LangChain model, and the AI responses are processed.

app > services > watsonx_service.py

```python
import os
import requests
import logging
from dotenv import load_dotenv
from app.services.code_generator import generate_code_snippet
from app.services.doc_generator import generate_docstring
from app.services.bug_resolver import resolve_bugs

load_dotenv()

# Watsonx API endpoint
WATSONX_API_URL = "https://eu-de.ml.cloud.ibm.com/ml/v1/text/generation?version=2023-05-29"

# Enable basic logging
logging.basicConfig(level=logging.INFO)

# Function to retrieve a Bearer token from IBM IAM using the API key
def get_iam_token() -> str:
    response = requests.post(
        url="https://iam.cloud.ibm.com/identity/token",
        headers={"Content-Type": "application/x-www-form-urlencoded"},
        data={
            "grant_type": "urn:ibm:params:oauth:grant-type:apikey",
            "apikey": os.getenv("API_KEY"),
        },
    )

    if response.status_code != 200:
        logging.error("Failed to get IAM token: %s", response.text)
```

app > routes > chat_routes.py

```python
# app/routes/chat_routes.py

from fastapi import APIRouter, Form, HTTPException
from app.langchain_module.conversation_handler import handle_conversation

router = APIRouter()

@router.post("/chat")
def chat_with_bot(message: str = Form(...)):
    try:
        response = handle_conversation(message)
        return {"response": response}
    except Exception as e:
        print(f"[❌ ERROR] {e}")
        raise HTTPException(status_code=500, detail="Failed to process message.")

```

Milestone3:main.pyDevelopment

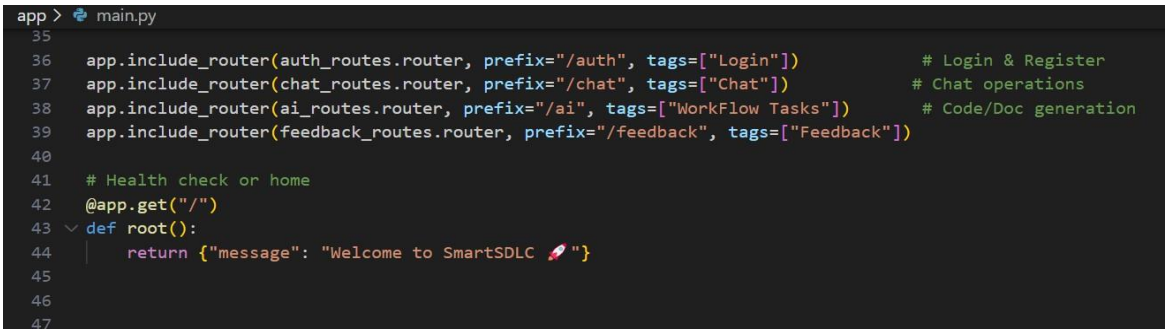
Milestone 3 focuses on creating and organizing the main control center of the application–themain.pyfile,whichpowerstheFastAPIbackendinSmartSDLC.This milestone involves linking each core SDLC functionality through clearly defined API

routes, ensuring input/output handling is reliable, and preparing the system for frontendinteraction.ThismodularsetupwillallowseamlessintegrationofWatsonx and LangChain responses while maintaining a clean API surface for the frontend.

Activity3.1:Writing the Main Application Logic in main.py

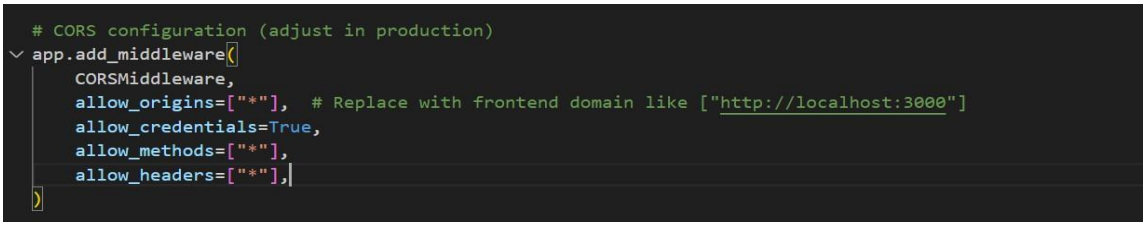
1:DefinetheCoreRoutesin main.py

- ImportandincluderoutersforeachoftheSmartSDLC'sfunctionalareas:requirement analysis, codegeneration, testing, summarization, bug fixing, authentication, chatbot, and feedback.
- UseFastAPI'sinclude_router()methodtomodularizetheroutedefinitionsand separateconcernsacrossfiles.
- Exampleroutermounts:
  - /ai:HandlesAI-basedendpointslikecodegeneration,testcreation,bugfixing.
  - /auth:Manageslogin,registration,anduservalidation.
  - /chat:PowersthefloatingchatbotviaLangChain.
  - /feedback:Handlessubmissionandstorageof feedback.



2:SetUp RouteHandlingand Middleware

FastAPImiddleware is configured using CORSMiddleware to ensure the frontend (e.g.,Streamlit)caninteractwiththebackendwithoutcross-originissues.Thisis especially useful during development and should be fine-tuned in production.



3:ImplementApplicationMetadataandRootRoute

TheFastAPIappiscreatedwithcustommetadatasuchasa**projecttitle,version**, and**description**tohelpidentifytheAPIoninteractiveSwaggerdocs(/docs).

```
# Local imports
from app.core.config import settings
from app.db.init_db import init_db
from app.routes import  chat_routes, ai_routes, auth_routes  , feedback_routes

app = FastAPI(
    title="SmartSDLC - AI-Enhanced Software Development Lifecycle",
    description="AI-powered microservice platform for accelerating SDLC using Watsonx, FastAPI, and GitHub in
    version="1.0.0"
)
```

Milestone4:FrontendDevelopment

Milestone4focusesoncreatinga**visuallyappealing**and**user-friendlyinterface**for SmartSDLCusing**Streamlit**.TheUIisdesignedtoprovidesmoothnavigationacross SDLCfunctionalitieslikerequirementclassification,codegeneration,bugfixing,and more.Thismilestoneensuresaresponsive,consistentexperiencewhileleveragingAI outputs effectively, and includes a built-in chatbot for dynamic user interaction.

Activity4.1:DesigningandDevelopingtheUser Interface

1:SetUptheBaseStreamlitStructure

- CreateamainHome.pyfilethat actsasthedashboardandentrypointof theapp.
- Addawelcomingherosectionwitha**Lottieanimation**,atitle,andatagline.
- Organizefeaturesinagridlayoutwithcleannavigationlinkstomodular pages.

2:Design aResponsive LayoutUsing Streamlit Components

- Usest.columns(),st.container(),st.markdown()forlayoutcontrolandconsistency.
- ApplycustomCSSstylingforbetterfonts,backgrounds,andcardshadows.
- Designaflexiblelayoutthatworkswellacrossdifferentscreensizesand resolutions.

3:CreateSeparatePagesforEachCoreFunctionality

- Buildfeature-specificmodulesunderthepages/directory:
  - Upload_and_Classify.py:UploadPDFandclassifyrequirements.
  - Code_Generator.py:Convertuserpromptsintoworkingcode.
  - Test_Generator.py:Generatetest cases.
  - Bug_Fixer.py:Automaticallyfixbuggycode.
  - Code_Summarizer.py:Summarizecodeintodocumentation.
  - Feedback.py:Collectuserfeedback.

- ConnecteachpagetocorrespondingFastAPIendpointsviaapi_client.py.

Activity 4.2: Creating Dynamic Interaction with Backend

1:IntegrateFastAPIwithStreamlitforReal-TimeContent

- Userequests.post()orrequests.get()insideapi_client.pytointeractwithbackendroutes like /ai/generate-code, /ai/upload-pdf, etc.
- Ensureuserinputslikeuploadedfilesorprompt textareformattedandpassedproperly.
- DisplayAIresponsesusingst.code(),st.success(),andmarkdownblocksforclarity.

2:EmbedaSmartFloatingChatbot

- AddaminimalinlinechatbotinHome.pyusingaStreamlitform.
- Usethe/chat/chat endpointtosendandreceivereal-timeresponses.
- Enhanceinteractionbyshowingemoji-based avatarsandsessionmemory.

Milestone5:Deployment

InMilestone5,thefocusisondeployingthe**SmartSDLCapplicationlocally**using FastAPIforthebackend and Streamlitforthefrontend.This involves settingup a secure environment, installing dependencies, connecting API keys for Watsonx, and launchingtheservicestosimulateareal-worldworkflow.Thismilestonehelpsensure that the app can run smoothly in a local setup before shifting to cloud platforms or CI/CDpipelines.

Activity5.1:PreparingtheApplicationforLocal

Deployment

1:Set Up a Virtual Environment

- CreateaPythonvirtualenvironmenttomanagedependenciesandavoidconflictswith other projects.
- Activatetheenvironmentandinstalldependencieslistedinrequirements.txttoensureall libraries (FastAPI, Streamlit, Watsonx SDK, etc.) are available.

2:ConfigureEnvironment Variables

- Setenvironmentvariablesforsensitivedatasuchas**IBMWatsonxAPIkey**, **modelIDs**,anddatabaseURLs.
- Createa.envfileinyourprojectroottosecurelystoreandloadthesesettingsduring runtime.
    - WATSONX_API_KEY=your_ibm_key_here
    - WATSONX_PROJECT_ID=your_project_id
    - WATSONX_MODEL_ID=granite-20b-code-instruct

Thesevaluesareloadedusingpython-dotenvinsideyourbackend(e.g., config.py).

Activity5.2:TestingandVerifyingLocal

Deployment

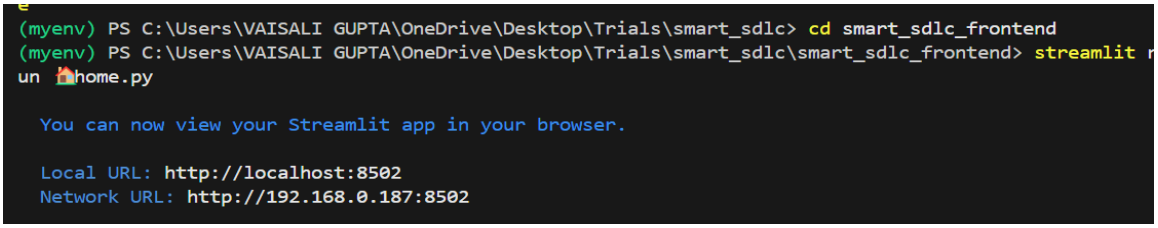
1:LaunchandAccesstheApplicationLocally

- **StarttheFastAPIbackend** usingUvicorn:uvicornapp.main:app--reload
- **RuntheStreamlitfrontend**: streamlit runfrontend/Home.py

Openyourbrowserandnavigateto:

- **StreamlitUI:**http://localhost:8502
- **FastAPISwaggerDocs:**http://127.0.0.1:8000/docs
- TesteachSmartSDLCfeature(requirementupload,codegeneration,bugfixing,chatbot, feedback) to ensure everything is connected and functioning correctly.

- **RuntheWeb Application**
- Nowtype"streamlitrun????home.py"command
- Navigatetothelocalhost whereyoucanview yourwebpage



Milestone6:Conclusion

The**SmartSDLC** platformrepresentsasignificantadvancementintheautomationof theSoftwareDevelopmentLifecyclebyintegrating**AI-poweredintelligence**into each phase—from requirement analysis to code generation, testing, bug fixing, and documentation.Byleveragingcutting-edgetechnologieslike**IBMWatsonx**,**FastAPI**, **LangChain**,and**Streamlit**,thesystemdemonstrateshowgenerativeAIcan streamline traditional software engineering tasks, reduce manual errors, and accelerate development timelines.

The platform's modular architecture and intuitive interface empower both technical and non-technical users to interact with SDLC tasks efficiently. Features such as requirement classificationfromPDFs,AI-generateduserstories,codegenerationfromnaturallanguage,auto test case generation, smart bug fixing, and integrated chat assistance illustrate the power of AI when applied thoughtfully within a development framework.

Overall,SmartSDLCnotonly improves productivityand accuracy butalsosets the foundation for future enhancements like CI/CD integration, team collaboration, versioncontrol,andclouddeployment.Itisasteptowardbuilding **intelligent, developer-friendlyecosystems**thatsupportmodernagiledevelopmentneedswith smart automation at its core.

Thewebapplicationwillopeninthewebbrowser:

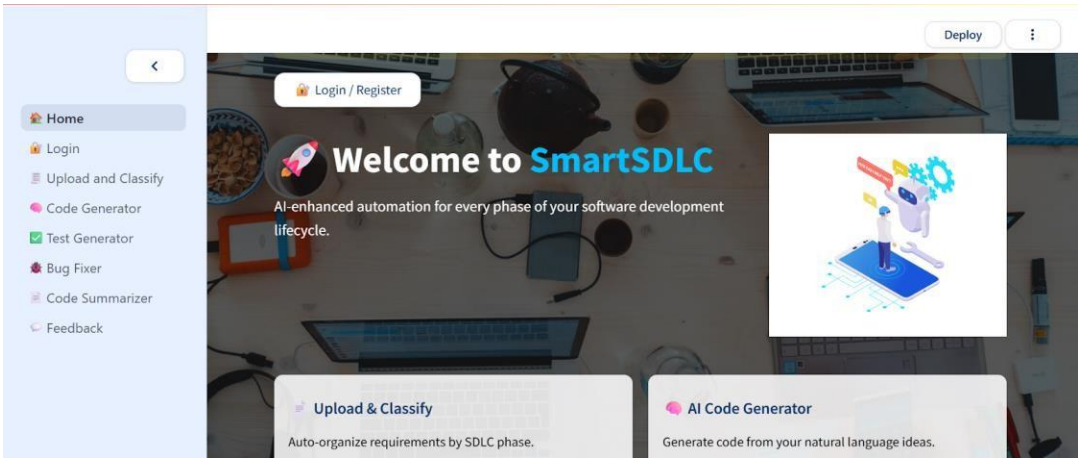


Figure3:"SmartSDLCDashboard–YourAI-poweredcommandcenterforacceleratingevery phase of the software development lifecycle."

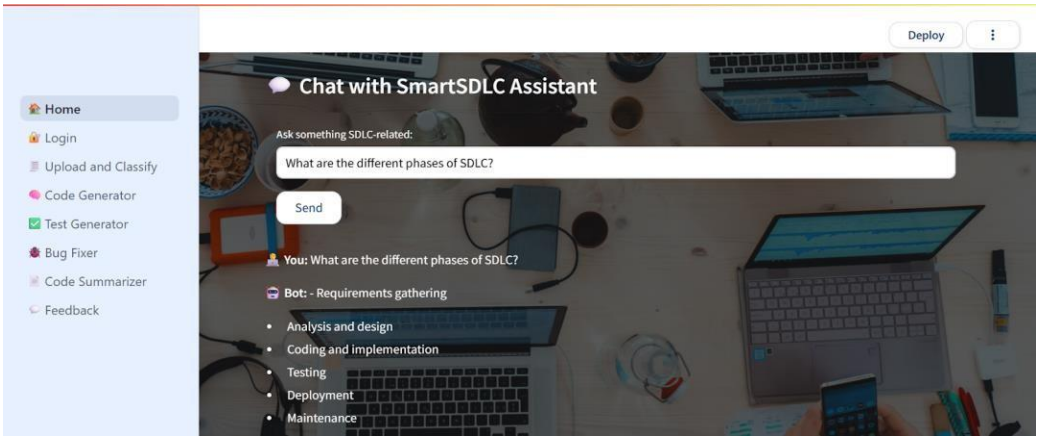


Figure4:"InteractiveChatwithSmartSDLCAssistant –InstantlylearnSDLCconceptsthrough AI-powered conversations."

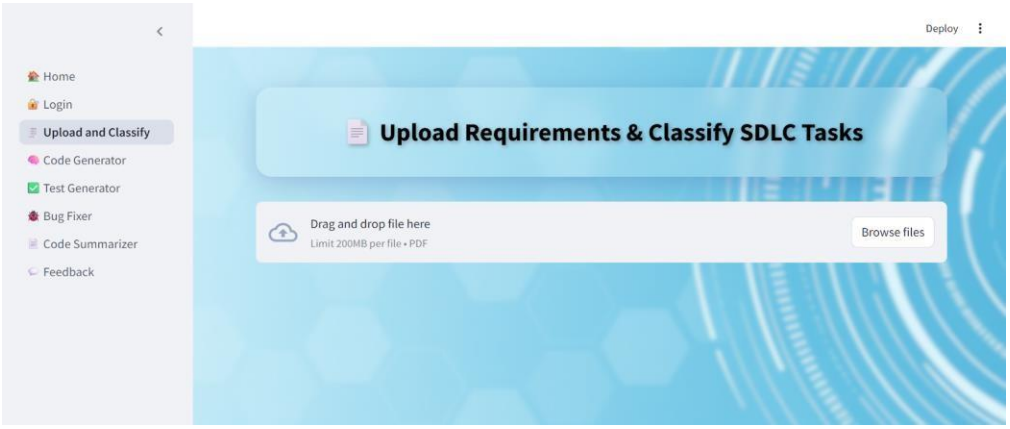


Figure5:Input:"UploadandClassify –EffortlesslyconvertrawPDF requirementsintostructured SDLC tasks using AI."

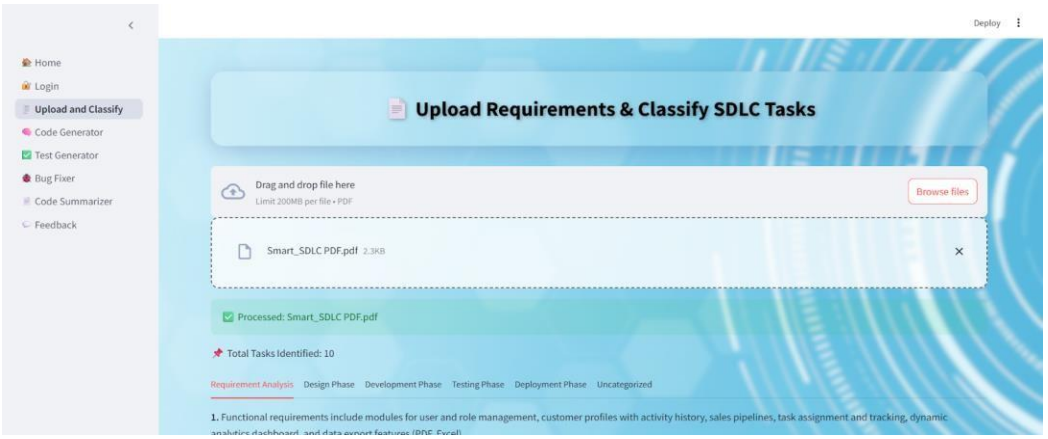


Figure6:Output:"SmartRequirementClassifier –Instantlyextractandcategorizesoftwaretasks into SDLC phases from your uploaded PDF."

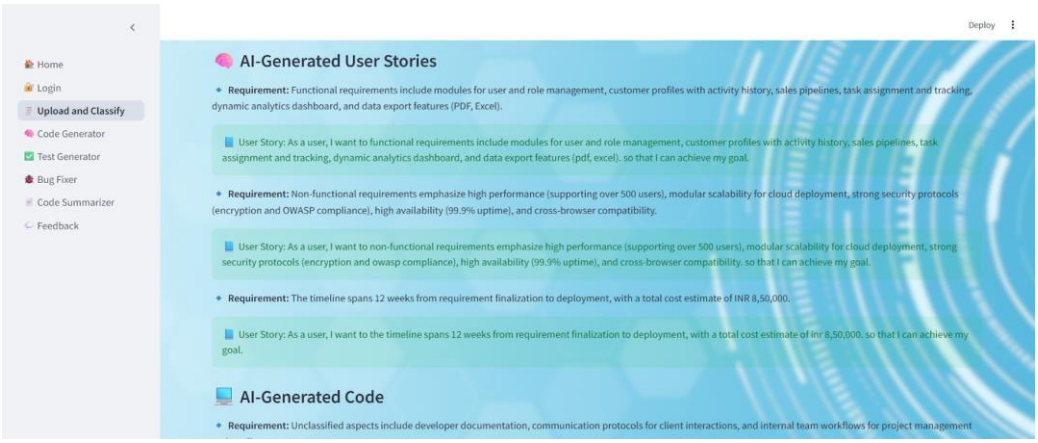


Figure7:"AI-GeneratedUserStories–Transformrawrequirementsintostructureduser-centric stories for agile development."

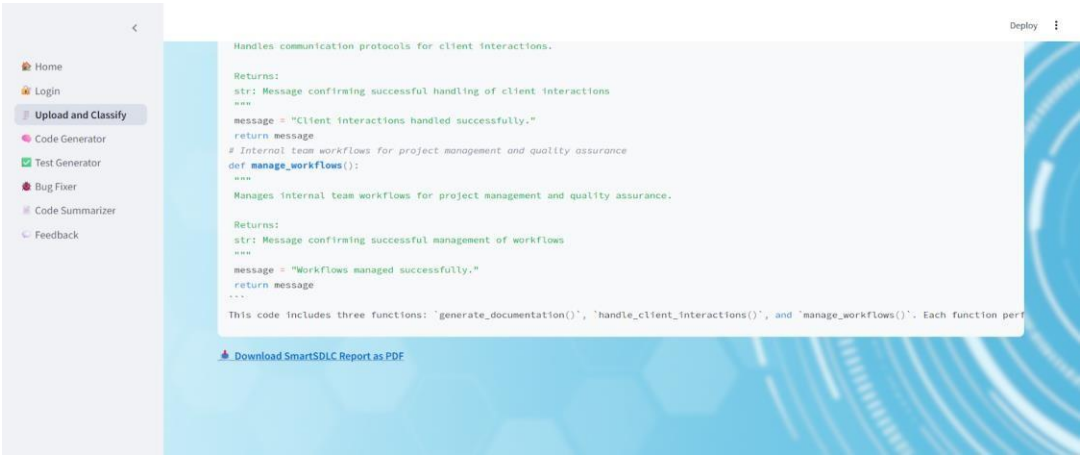


Figure8:"Auto-GeneratedSummarywithDownloadableInsights –ReviewAI-explainedcode and export the SmartSDLC report as a professional PDF."

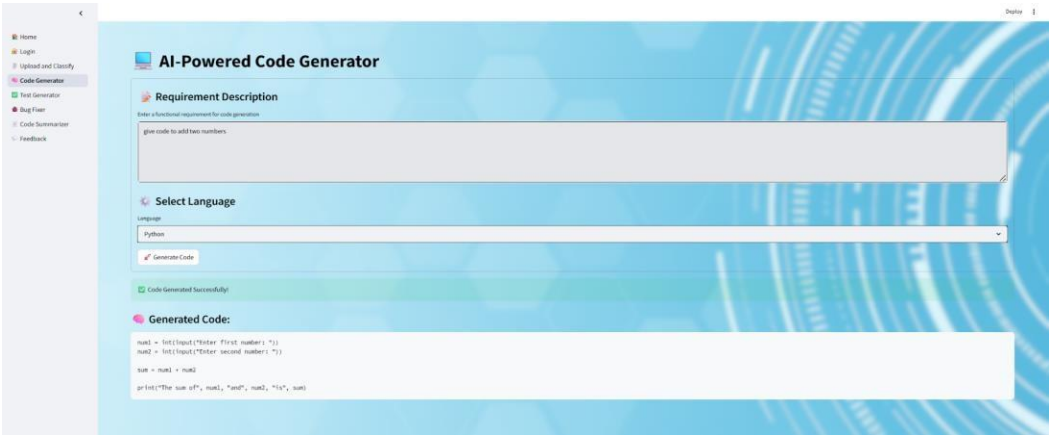


Figure9:"AICodeGenerator–Instantlyconvertnaturallanguagerequirementsintoclean, executable code with just one click."

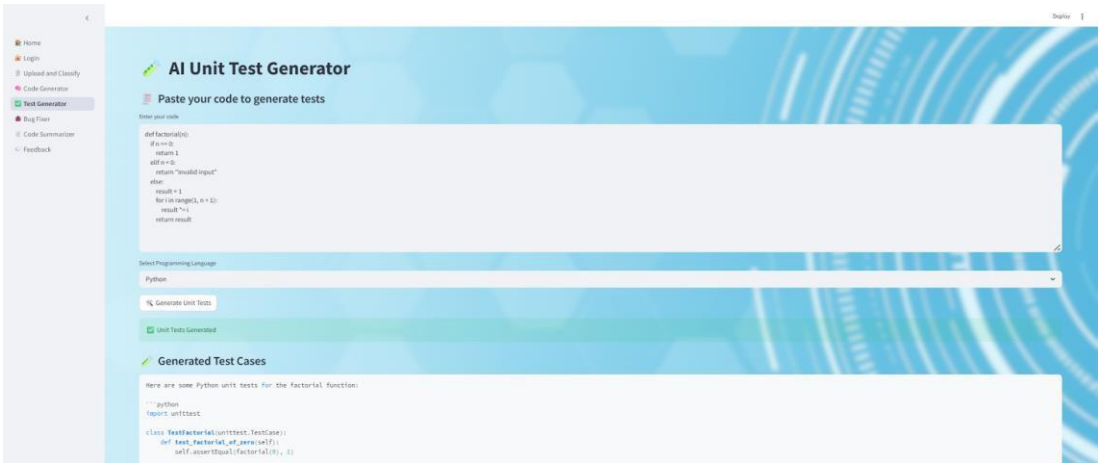


Figure10: AI-poweredtooltoauto-generateunit testsfromuser-providedcode.

Figure11:"AI-PoweredBugFixer–Instantlydetectandcorrectcodeissuesforcleaner,error- free execution."

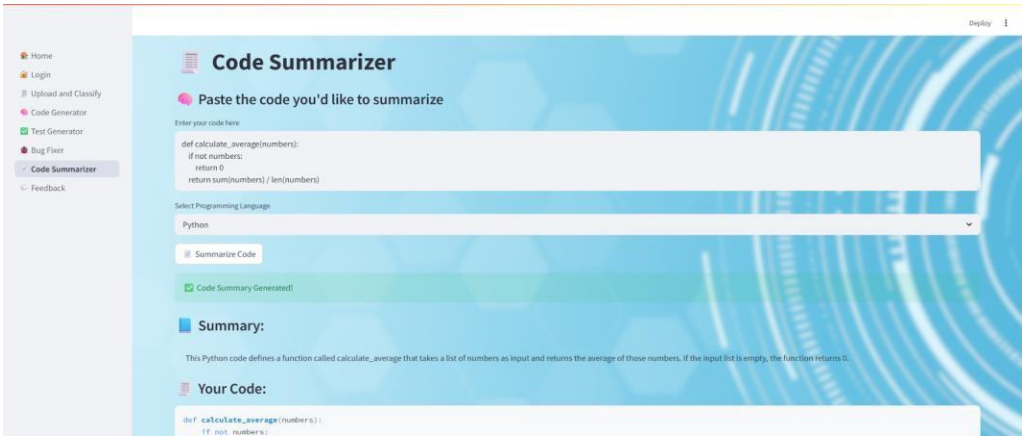


Figure12:“AI-poweredCodeSummarizerinterfacedisplayingagenerated summaryforthe providedcodesnippet.”

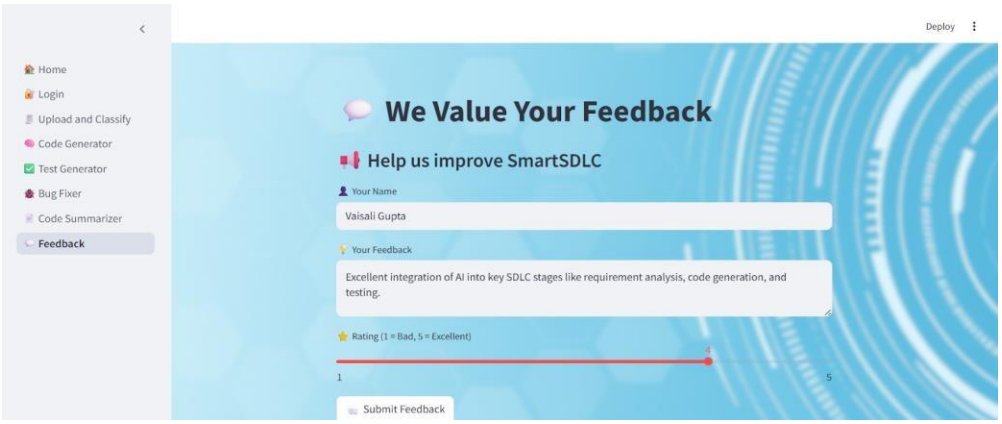


Figure13:"UsersubmittingfeedbackontheSmartSDLCsystem,highlightingeffectiveAI integration across SDLC phases"

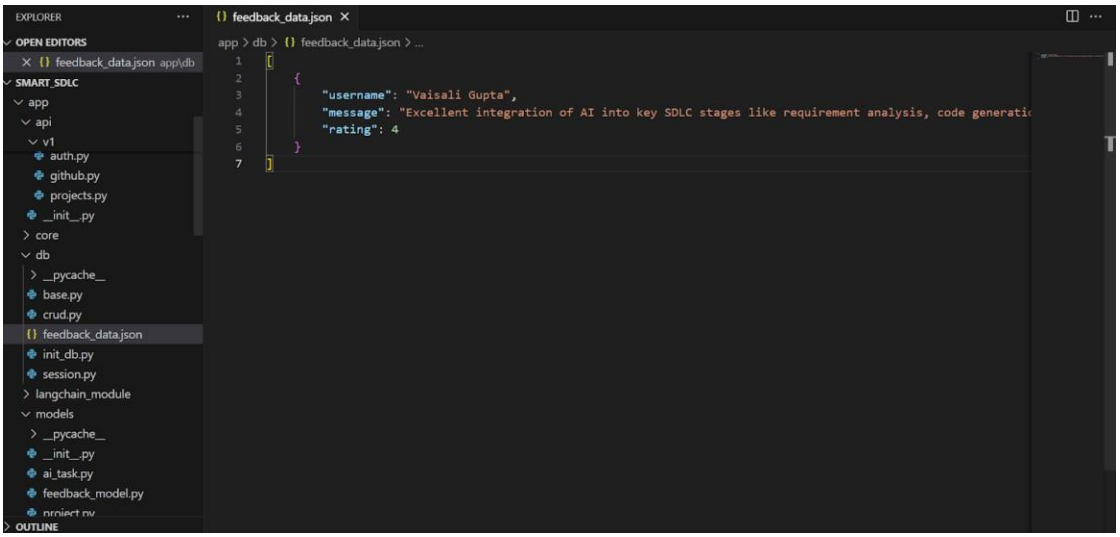


Figure14:Userfeedbackdatastoredinfeedback_data.jsonfilewithintheSmartSDLC application's database module.

TechStack

- **Frontend**:React.js/Next.js
- **Backend**:Python(FastAPI)or Node.js(Express)
- **AIModels**:OpenAIGPT/BERT/CustomML models
- **Database**:MongoDB orPostgreSQL
- **CI/CD**:GitHubActions,Docker
- **Authentication**:JWT-basedAuth
- **Testing**:Jest,PyTest,Postman/Newman

⚙️ 3.SetupInstructions

Prerequisites

- Node.js(iffrontend/backendisinJS)
- Python3.9+(for AIandbackend)
- Docker&DockerCompose(optional)
- MongoDBorPostgreSQLinstalled
- `.env`filewithenvironmentvariables

InstallationSteps

```
#Clonetherepository
gitclonehttps://github.com/your-org/smartsdlc.gitcd
smartsdlc

#BackendSetup cd
backend
pipinstall-r requirements.txt

#FrontendSetup
cd ../frontend
npm install
```

AIModelSetup(optionalifnotusingAPI-basedLLMs)

- DownloadmodelsorconnectOpenAI/GPTAPIs
- SetAPIkeysin `.env`files

🗂 4.FolderStructure

```
smartsdlc/
│
├──backend/
│   ├──app/
│   │   ├──api/                  #FastAPIroutes
│   │   ├──services/             #Businesslogic,AImodules
│   │   ├──models/               #PydanticmodelsorORM
│   │   ├──utils/                #Helperfunctions
│   │   └──main.py               #Appentrypoint
│   ├──tests/                    #Backendtests
│   └──requirements.txt
```

```
│
├── frontend/
│   ├──components/               # ReusableUIcomponents
│   ├──pages/                    # Pages(Next.jsstructure)
│   ├──services/                 # APIcalls
│   ├──styles/
│   └──app.js/ main.jsx
│
├── docs/                        # APIdocs,architecture diagrams
├── docker-compose.yml
├── README.md
└── .env.example
```

▶️5.RunningtheApplication

StartBackend

```
cdbackend
uvicornapp.main:app--reload
```

Or,ifusingNode.js:

```
cdbackend
npm start
```

StartFrontend

```
cdfrontend
npmrundev
```

Docker (optional)

```
docker-composeup--build
```

📘6.APIDocumentation

APIdocsareauto-generatedusing**Swagger** (ifFastAPI)or**Postman Collection**.

Swagger(FastAPI)

- Visit:http://localhost:8000/docs

SampleEndpoints

Method	Endpoint	Description
POST	/api/requirements/analyze	Extractsrequirementsfrom text
POST	/api/design/generate	Generatesarchitecture design
POST	/api/code/generate	Generatescodefromdesign
POST	/api/test/generate	Generatestestcases
GET	/api/monitor/status	Monitorsdeployedapp

7.Authentication

Uses**JWT-basedAuth** toprotect routes.

AuthFlow

1. Usersignsuporlogsin.
2. BackendissuesJWTtoken.
3. Tokenisstoredinbrowser(cookiesor localStorage).
4. TokenissentinAuthorizationheaderforsecureAPI calls.

```
Authorization:Bearer<your_token_here>
```

8.UserInterface

FrontendFeatures

- Dashboardformanagingprojects
- RequirementinputUI(textorvoice)
- Visualdesigngenerator(UMLdiagrams)
- AI-generatedcodeviewer
- Testcase editor
- Deploymentstatus+logsviewer
- Login/RegisterwithJWTAuth

Tools

- React+TailwindCSS
- Chart.js/Mermaid.jsforvisualdiagrams
- Monacoeditorfor codedisplay

🧪 9.Testing

BackendTesting

- **PyTest**or**unittest**
- TestAPIroutes,services,andAI modules

```
cdbackend
pytest
```

FrontendTesting

- **Jest**+**ReactTesting Library**

```
cdfrontend
npm test
```

APITesting

- **Postman**or**Newman**scriptstotestendpoints
- Includedin`/docs/api/`folder

✅Summary

Feature	Description
AI Assistance	IntegratedAIateachSDLC phase
Full-stackApp	Frontend+Backend+DB+AI
APIDocs	Auto-generatedwithSwagger
Auth	JWT-securedAPIaccess
Testing	Unit+Integration+API