**Project Design Phase**

**Solution Architecture**

| Date | 25 JUNE 2025 |
| --- | --- |
| Team ID | LTVIP2025TMID21159 |
| Project Name | SmartSDLC – AI-Powered Software Development Lifecycle Optimization |
| Maximum Marks | 2 Marks |

### ****Purpose of the Architecture****

The **SmartSDLC** solution architecture bridges the gap between common software development inefficiencies and AI-powered automation. It defines the structural flow, components, and technologies that collaboratively deliver an intelligent, scalable, and user-friendly platform.

### ****Key Objectives****

* Identify the best AI/NLP-based solution to automate key SDLC stages.
* Present a modular and interpretable architecture to all stakeholders.
* Define technical layers and component responsibilities clearly.
* Enable easy extension, maintenance, and deployment of the solution.

**Example - Solution Architecture Diagram:**



*Figure 1: Architecture and data flow of the voice patient diary sample application*

**Reference:** [**https://aws.amazon.com/blogs/industries/voice-applications-in-clinical-research-powered-by-ai-on-aws-part-1-architecture-and-design-considerations/**](https://aws.amazon.com/blogs/industries/voice-applications-in-clinical-research-powered-by-ai-on-aws-part-1-architecture-and-design-considerations/)

**Core Architectural Components**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Component** | | |  | | --- | | **Technology Used** | | |  | | --- | | **Purpose** | |
| **Frontend UI** | |  | | --- | | Streamlit, streamlit-lottie |  |  | | --- | |  | | User interface for text input, PDF upload, and feature access |
| **Backend API Layer** | |  | | --- | | FastAPI, uvicorn, requests |  |  | | --- | |  | | Handles routing, model invocation, fallback logic, and concurrency |
| **Document Parser** | |  | | --- | | PyMuPDF |  |  | | --- | |  | | Extracts and structures requirements from uploaded PDFs |
| **AI Engine** | |  | | --- | | transformers, torch, huggingface\_hub |  |  | | --- | |  | | Core LLM logic for classification, generation, summarization |
| **Environment Manager** | |  | | --- | | dotenv |  |  | | --- | |  | | Secure token management for Hugging Face and fallback (Groq) APIs |
| **Async Task Manager** | |  | | --- | | asyncio.create\_task() |  |  | | --- | |  | | Enables concurrent execution of API/model requests |
| **Fallback Support** | |  | | --- | | Groq API |  |  | | --- | |  | | Ensures model inference continues if local LLM fails or GPU is unavailable |

### ****Module-Wise Workflow****

1. **PDF/Requirement Input**  
   ➤ User uploads PDF → pdf\_parser.py → text extraction → requirement classification via generate()
2. **Code & Test Case Generation**  
   ➤ Text or prompt given → routed via FastAPI → model in ai\_engine.py → returns code/test cases
3. **Bug Fixing & Summarization**  
   ➤ User provides buggy or unstructured code → model processes and responds with fixed code/summaries
4. **Chatbot**  
   ➤ Natural language SDLC queries routed to chatbot.py → responded using the same generate() logic

**System Flow Diagram**

[ User Interface (Streamlit) ]

↓

[ FastAPI Backend Layer ]

↓

┌────────────┬─────────────┬─────────────┐

│ pdf\_parser │ code\_gen.py │ summarizer │

│ bug\_fixer │ chatbot.py │ test\_cases │

└────────────┴─────────────┴─────────────┘

↓

[ AI Engine (IBM Granite via Hugging Face) ]

↓ ↘ (if fail) ↘

[ Output ] [ Groq API Fallback ]

### ****Development Phases****

1. **Phase 1** – Core module implementation (code generation, test generation, bug fixing)
2. **Phase 2** – UI integration with backend and basic error handling
3. **Phase 3** – Chatbot module and PDF parser integration
4. **Phase 4** – Fallback, logging, and performance optimization
5. **Phase 5** – Deployment, testing, and final documentation