Project Design Phase
Solution Architecture

Date	15 February 2025
Team ID	LTVIP2025TMID36326
Project Name	Smart SDLC AI Assistant
Maximum Marks	4 Marks

1 Solution Architecture

1.1 Purpose

Solution architecture is a complex process with many sub-processes that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

1.2 Architecture Description

The Smart SDLC AI Assistant is a web-based platform designed to automate Software Development Life Cycle (SDLC) tasks, addressing the challenges of time-consuming manual processes and error-prone coding for developers, students, and small teams. The solution leverages a micro-services architecture with the following components:

- **Frontend**: Built with React and Tailwind CSS, providing a dark-themed, intuitive UI for inputting requirements, editing code, viewing test cases, and interacting with the chatbot.
- **Backend**: Utilizes Node.js with Express.js for API management and FastAPI for high-performance AI-driven endpoints, handling requests for requirement analysis, code generation, and testing.
- **AI Engine**: Integrates pre-trained large language models (LLMs) via xAI's API (https://x.ai/api) for tasks like requirement structuring, code generation, bug detection, and summarization.

- **Database**: MongoDB stores user inputs, generated code, test cases, and summaries, with schemas for user profiles, project requirements, and outputs. SQLite is used for local development.
- **Infrastructure**: Deployed on Kaggle for cloud scalability, with local deployment support in VS Code. Load balancers ensure high availability (99.9% uptime).

1.3 Data Flow

1. **User Input**: Developers input plain text requirements, code snippets, or queries via the React frontend.
2. **API Processing**: The backend (Node.js/Express.js, FastAPI) routes requests to the AI engine for processing (e.g., structuring requirements, generating code).
3. **AI Inference**: The AI engine, powered by xAI's API, processes inputs and generates outputs (e.g., structured modules, code, test cases).
4. **Data Storage**: Results are stored in MongoDB for persistence and retrieval, with SQLite for local testing.
5. **Output Delivery**: The frontend displays generated outputs (e.g., code, test cases, summaries) to the user, with real-time editing capabilities.

1.4 Features and Development Phases

- **Features**: Requirement analysis, multilingual code generation (Python, Java, C++), automated test case creation, bug detection and fixing, code summarization, and a chatbot for SDLC queries.
- **Phases**:
  - **Sprint 1**: Requirement analysis and initial code generation (10 story points).
  - **Sprint 2**: Code editing, test case generation, bug fixing (11 story points).
  - **Sprint 3**: Code summarization, chatbot integration (4 story points).

1.5 Solution Architecture Diagram

[Placeholder for architecture diagram: Illustrates data flow from user input through frontend (React), backend (Node.js/FastAPI), AI engine (xAI API), and MongoDB storage, with arrows indicating request and response paths.]

Reference:

https://aws.amazon.com/blogs/industries/voice-applications-in-clinical-researchpowered-by-ai-on-aws-part-1-architecture-and-design-considerations/