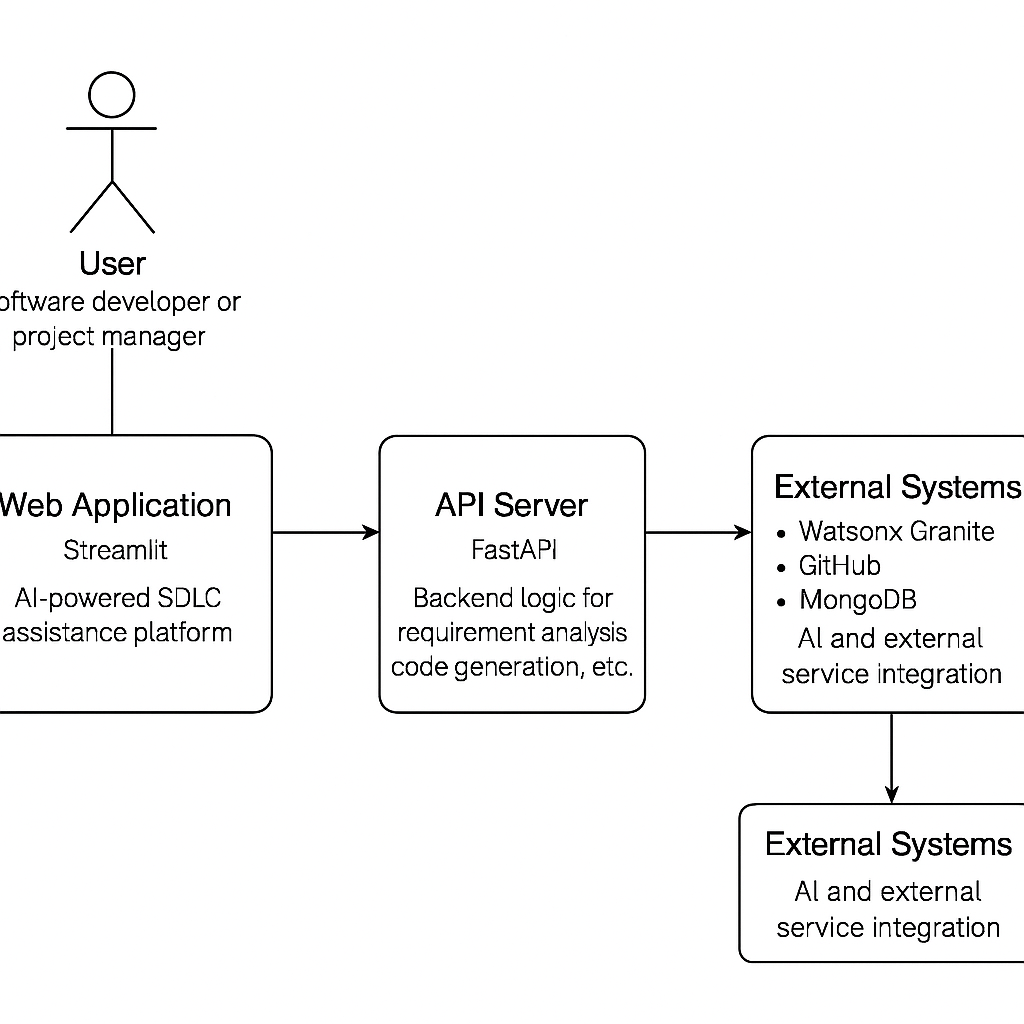
Project Design Phase-II

Technology Stack (Architecture & Stack)

|  |  |
| --- | --- |
| Date | 31 January 2025 |
| Team ID | LTVIP2025TMID32102 |
| Project Name | SmartSDLC – AI-Enhanced Software Development Lifecycle |
| Maximum Marks | 4 Marks |

Technical Architecture:****

This AI-powered system processes software project requirements, performs classification, generates code, creates documentation, and offers bug detection using Watsonx and other generative AI services. Below are component-wise mappings.

**Guidelines:**  
 1. Actors / Users (System Context)

* User: Represented with a stick figure labeled "Software Developer or Project Manager".
* Guideline: Clearly identify roles that interact with the system (e.g., Mobile App user, Admin, Developer).

#### 2. Web Application (Container)

* Label it as a Streamlit Web Interface.
* Describe its purpose: “User-facing interface for uploading requirements, viewing outputs, and interacting with the SmartSDLC tools.”
* Mention technologies used (Streamlit, HTML, etc.)

#### 3. API Server (Container)

* Label it FastAPI backend.
* List responsibilities like:
  + Requirement classification
  + Code/test generation
  + Calling AI models and services
* Mention optional use of async API endpoints for better performance.

#### 4. External Systems

* Identify AI services and databases:
  + Watsonx Granite for prompt-to-code generation
  + GitHub API for code commits
  + MongoDB for storing structured project data

5. Best Practices

* Keep it simple, 3–5 main components.
* Use clear labels and arrows for communication flow.
* Add icons or color if converting to presentation.

Table-1:Components & Technology

| S.No | Component | Description | Technology |
| --- | --- | --- | --- |
| 1. | User Interface | Web interface for uploading requirements, viewing output, and interacting with AI assistant | Streamlit, HTML5, CSS3, JavaScript |
| 2. | Application Logic-1 | Requirement analysis & classification | Python (FastAPI) |
| 3. | Application Logic-2 | Code generation & documentation assistant | IBM Watsonx Granite 13B Instruct |
| 4. | Application Logic-3 | Chatbot for query answering and guidance | IBM Watson Assistant, OpenAI GPT API (optional) |
| 5. | Database | Store project metadata, user logs, and results | MongoDB Atlas / SQLite |
| 6. | Cloud Database | Scalable cloud-hosted DB for team data | IBM Cloudant / Firebase |
| 7. | File Storage | Store uploaded PDFs, result files | IBM Cloud Object Storage / Google Drive API |
| 8. | External API-1 | GitHub Integration (e.g., commit code, fetch issues) | GitHub REST API |
| 9. | External API-2 | Email service for notifications | SendGrid API / SMTP |
| 10. | Machine Learning Model | AI requirement classifier, code generator | Watsonx Granite, custom Transformers via Hugging Face |
| 11. | Infrastructure | Deployment platform | IBM Cloud Foundry / Google Colab / Streamlit Cloud / Kubernetes (for scale) |

Table-2:Application Characteristics

| S.No | Characteristic | Description | Technology |
| --- | --- | --- | --- |
| 1. | Open-Source Frameworks | Core frameworks for web, ML, and deployment | FastAPI, Streamlit, PyTorch, Transformers (HF) |
| 2. | Security Implementations | Secure APIs, authentication, file handling | OAuth 2.0, SHA-256, IAM Roles, HTTPS, Streamlit Auth |
| 3. | Scalable Architecture | Supports modular services, scalable on cloud | Microservices pattern using FastAPI & containerization |
| 4. | Availability | 99.5% uptime with fallback mechanisms | Streamlit Cloud + IBM Cloud Load Balancing |
| 5. | Performance | Caching, async APIs, optimized model calls | Redis (optional), CDN, async FastAPI endpoints |