**Project Design Phase-II**

**Technology Stack (Architecture & Stack)**

|  |  |
| --- | --- |
| Date | 31 January 3035 |
| Team ID | LTVIP2025TMID32163 |
| Project Name | Smart SDLC – AI Enhanced Software Development Lifecycle |
| Maximum Marks | 4 Marks |

**Technical Architecture:**

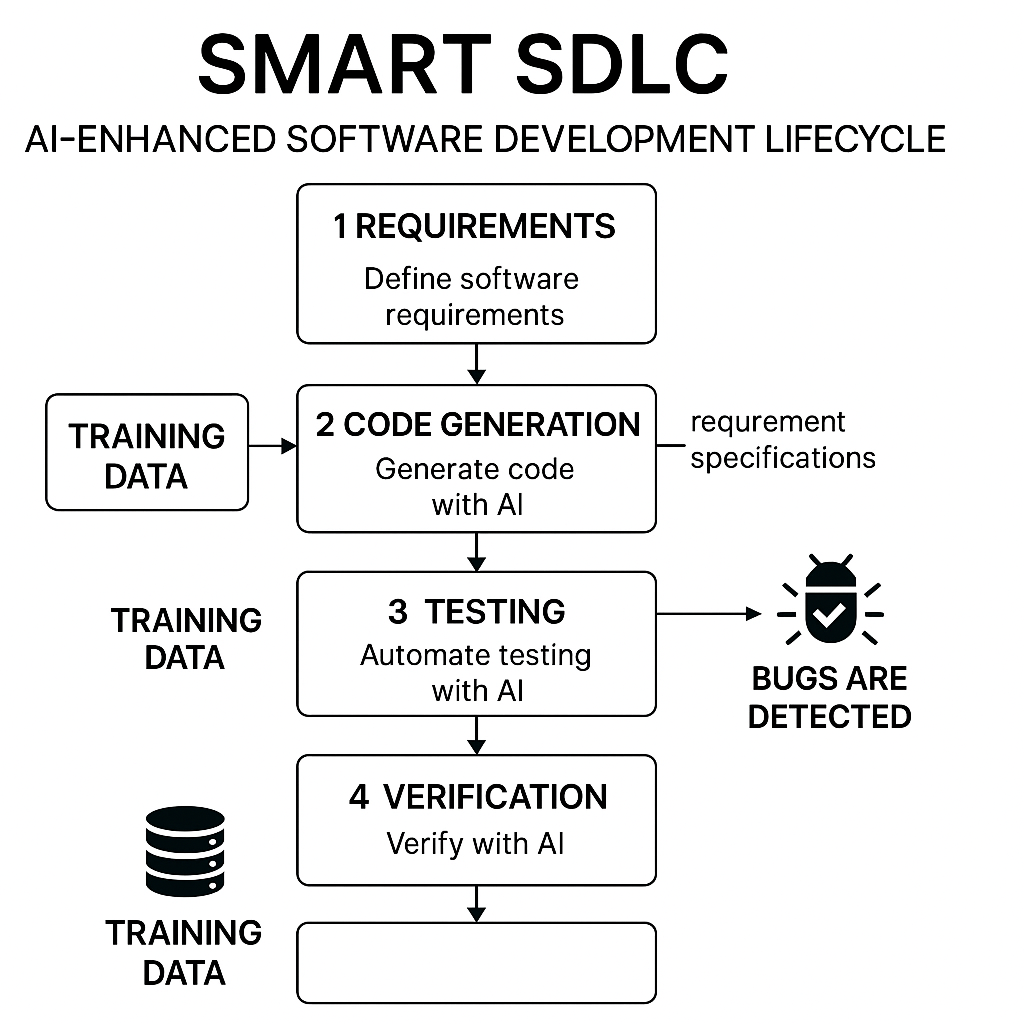
The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

**Example: Order processing during pandemics for offline mode**

**Reference:** [**https://developer.ibm.com/patterns/ai-powered-backend-system-for-order-processing-during-pandemics/**](https://developer.ibm.com/patterns/ai-powered-backend-system-for-order-processing-during-pandemics/)

### ****Guidelines: Smart SDLC – AI Enhanced Software Development Lifecycle****

* **Include all the processes (As an application logic / Technology Block):**
  + User Interface – React JS
  + Backend API Engine – FastAPI (Python)
  + Prompt Generator – Custom Python Logic
  + AI Integration Layer – IBM Granite-3.3-2b-instruct API
  + Authentication – OAuth 2.0 / JWT
  + Response Formatter – Python Formatter
  + Logging & History – Python Logging + MongoDB
* **Provide infrastructural demarcation (Local / Cloud):**
  + UI hosted on **Cloud** (Vercel / Netlify / IBM Cloud)
  + Backend deployed on **Cloud or Local** (Render / IBM Cloud Foundry / Docker)
  + AI Model accessed from **IBM Cloud**
  + Database hosted on **Cloud** (IBM Cloudant / Firebase) or **Local** (MongoDB / SQLite)
* **Indicate external interfaces (third party API’s etc.):**
  + IBM Granite AI Model API
  + Google / GitHub OAuth API for login
  + GitHub API (optional for project integration)
* **Indicate Data Storage components / services:**
  + MongoDB / Firebase for user data and session storage
  + IBM Cloud Object Storage / Local Filesystem for logs and files
  + Redis (optional) for caching AI responses
* **Indicate interface to machine learning models (if applicable):**
  + Prompts are sent to IBM Granite-3.3-2b-instruct via secured API
  + AI responses are processed and returned to the user via FastAPI



Certainly! Below are **Table-1 (Components & Technologies)** and **Table-2 (Application Characteristics)** specifically tailored for the **Smart SDLC** application.

### ✅ ****Table-1: Components & Technologies****

| **S.No** | **Component** | **Description** | **Technology** |
| --- | --- | --- | --- |
| 1. | User Interface | Web-based UI for interaction with SDLC assistant | HTML, CSS, JavaScript, React JS |
| 2. | Application Logic-1 | Core API handling and routing logic | Python, FastAPI |
| 3. | Application Logic-2 | AI prompt generation and request formatting | Custom Python Modules |
| 4. | Application Logic-3 | Integration with generative AI model | IBM Granite-3.3-2b-instruct Model (via API) |
| 5. | Database | Store user sessions or interactions (optional) | MongoDB / SQLite |
| 6. | Cloud Database | Scalable cloud-based DB for persistent data (optional) | IBM Cloudant / Firebase Firestore |
| 7. | File Storage | Store logs, user uploads, model outputs (if needed) | IBM Cloud Object Storage / Local File System |
| 8. | External API-1 | Project metadata enrichment (optional feature) | GitHub API (for repo integration) |
| 9. | External API-2 | Authentication via Google/GitHub | OAuth2 APIs |
| 10. | Machine Learning Model | AI model used for contextual SDLC assistance | IBM Granite-3.3-2b-instruct (Generative LLM) |
| 11. | Infrastructure | Hosting and deployment environment | IBM Cloud Foundry, Docker, Render, or Kubernetes Cluster |

### ✅ ****Table-2: Application Characteristics****

| **S.No** | **Characteristics** | **Description** | **Technology / Implementation** |
| --- | --- | --- | --- |
| 1. | Open-Source Frameworks | Core backend and frontend frameworks used | FastAPI (Python), React JS, MongoDB |
| 2. | Security Implementations | Securing APIs and data with modern standards | OAuth2, JWT Authentication, HTTPS, Input Validation |
| 3. | Scalable Architecture | Modular & extensible architecture for scale-out scenarios | Microservices (FastAPI modules), Docker, Kubernetes |
| 4. | Availability | Ensures continuous uptime and access | Load Balancers, Cloud Auto-Scaling, Health Monitoring Tools |
| 5. | Performance | Optimized for high responsiveness and concurrent requests | Caching (Redis), Async I/O (FastAPI), CDN (if static files) |

**References:**

[**https://c4model.com/**](https://c4model.com/)

[**https://developer.ibm.com/patterns/online-order-processing-system-during-pandemic/**](https://developer.ibm.com/patterns/online-order-processing-system-during-pandemic/)

[**https://www.ibm.com/cloud/architecture**](https://www.ibm.com/cloud/architecture)

[**https://aws.amazon.com/architecture**](https://aws.amazon.com/architecture)

[**https://medium.com/the-internal-startup/how-to-draw-useful-technical-architecture-diagrams-2d20c9fda90d**](https://medium.com/the-internal-startup/how-to-draw-useful-technical-architecture-diagrams-2d20c9fda90d)