**Learning Management System (LMS)**

**(Pattern- Client-Server)**

**1. Introduction**

This project implements a lightweight Learning Management System (LMS) using Python. It enables role-based access for students and instructors to register, authenticate, and interact with learning resources. The system comprises a **client-server architecture** integrating GUI components, socket-based communication, and a persistent SQLite database.

**2. System Architecture**

**2.1 Architectural Pattern**

The system follows a Client-Server Architectural Pattern, where the client application (Tkinter-based GUI) communicates with a centralized server over TCP/IP sockets. The server manages business logic, handles database operations, and ensures role-based access to system resources. This design provides modularity, central control, and the capability to handle multiple concurrent client connections using threading.

**2.2 Components Overview**

1. **Client Interface:** Built using Tkinter for graphical user interactions.
2. **Server Backend:** Manages client requests, authentication, and data transactions.
3. **Database Layer:** SQLite database for storing user accounts, course data, and learning resource URLs.

**System Architecture and Workflow**

This system follows a **client-server model**. The architecture is composed of the following primary components:

* **Client Application (GUI):** A Tkinter-based desktop interface enabling users to interact with the LMS. It handles user registration, login, course selection, and displays course-specific resources.
* **Server Application:** A Python-based multi-threaded server handling client requests, managing database operations, and responding to user actions like registration, authentication, and resource queries.
* **Database (SQLite):** A lightweight relational database storing user data, course information, and associated learning resources. It serves as the backend storage engine for the server.

**Technical Workflow:**

1. The **User** (Student or Instructor) interacts with the **Client GUI**.
2. The **Client** establishes a socket connection with the **Server**.
3. The **Client** sends structured requests (e.g., login, registration, resource upload, or retrieval) to the **Server**.
4. The **Server** processes these requests by interacting with the **SQLite database**.
5. Processed results (success, error messages, or retrieved data) are sent back to the **Client GUI** for display.



**System Architecture Diagram**

**4. Technologies Used**

* **Python –** Core language for client and server logic.
* **Tkinter –** GUI framework for user-facing interfaces.
* **Socket –** TCP-based communication for client-server interactions.
* **Threading –** Enables concurrent handling of multiple client sessions.
* **SQLite –** Persistent, lightweight relational database for storing data.

**5. Conclusion**

This Learning Management System demonstrates a distributed application architecture leveraging Python’s networking, GUI, and database libraries. It successfully integrates client-server communication, concurrency management, and persistent data storage to simulate a functional educational platform. The project highlights essential concepts in software engineering such as modular design, architectural pattern implementation, and data handling.