Learning Management System (LMS)

(Pattern- Publisher Subscriber)

1. Introduction

This project implements a real-time Learning Management System (LMS) using Python and the Publisher-Subscriber (Pub/Sub) pattern. It enables event-driven communication using Redis channels to support asynchronous updates between system components. The LMS provides role-based access for students and instructors, enabling users to register, subscribe to courses, manage learning resources, and receive real-time updates.

The architecture supports **true real-time** notifications, making it ideal for scenarios where immediate visibility of updates (like announcements, course changes, or new resources) is essential.

2. System Architecture

2.1 Architectural Pattern

The system is based on the **Publisher-Subscriber** design, where:

- **Publishers** broadcast events (e.g., new user, new announcement, subscription).
- **Subscribers** listen to specific channels and react to updates instantly.

Redis Pub/Sub is used to facilitate message distribution. This decouples components, allowing them to operate independently while staying synchronized through published events.

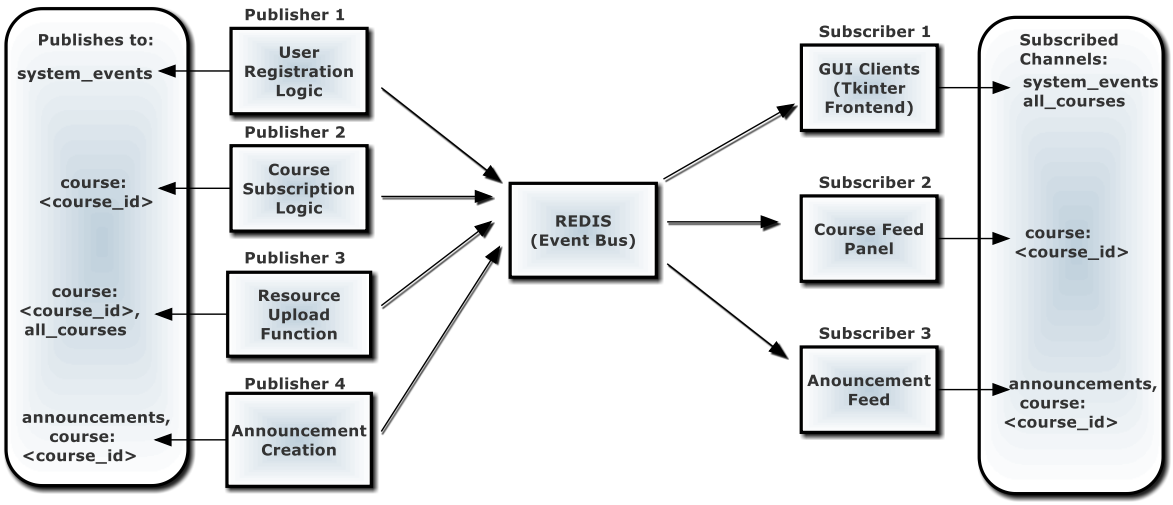

Fig. 01: System Architecture Pattern

2.2 Components Overview

1. **Graphical User Interface (GUI)**

- Built using Tkinter.
- Subscribes to Redis channels (e.g., course:<id>, system_events) on user login.

- Receives real-time updates via Redis listeners and displays them in the UI (Activity Feed).

2. **Event Processor (Publisher)**

- Publishes structured events to Redis when certain actions are performed, such as:
    - User registration
    - Course subscriptions
    - Resource uploads
    - Announcements

3. **Redis (Pub/Sub Broker)**

- Manages message broadcasting and event distribution.
- Allows multiple subscribers to listen to the same channel for real-time updates.

4. **SQLite Database**

- Stores persistent data (users, courses, resources, subscriptions, announcements).
- Used alongside event publishing to maintain consistency and long-term data storage.

3. System Architecture and Workflow

1. Instructor uploads a new resource to a course.
2. The server publishes a notification to a Redis channel like course:CS101.
3. Redis delivers the message to all clients (students) subscribed to that course.
4. When a student accesses their dashboard, the system checks for new resources based on the last time they viewed the course.
5. Only new resources are shown, creating a dynamic and efficient notification mechanism.

3. Real-Time Event Flow

3.1 Event-Driven Workflow

1. A user performs an action (e.g., uploads a resource).
2. The system publishes an event to a Redis channel:
    - Example: A new_resource event is published to course:CS101 and all_courses.
3. Subscribed GUI clients receive the event in real time.
4. The UI updates the Activity Feed with the event content.

4. Technologies Used

- **Python –** Core programming language.
- **Redis (Pub/Sub) –** Real-time event distribution and decoupled messaging.

- **Tkinter –** GUI framework for interactive desktop interfaces.
- **SQLite –** Persistent local storage for system data.
- **Threading –** Used to handle background listeners for Redis in the client.

5. Conclusion

The project demonstrates a clean implementation of the Publisher-Subscriber design pattern within an LMS context. It showcases how publishers (instructors) can broadcast updates without direct connection to subscribers, and how subscribers (students) can passively receive timely content. This results in a scalable, decoupled, and responsive system for modern learning platforms.