**USER AUTHENTICATION MODULE OF EDULEARN**

**Proposed Modifications to Improve Cohesion and Reduce Coupling**

1. **Separation of Concerns**:
   * Split the **User Authentication Service** into smaller, more focused services:
     + **Login Service**: Handles only login operations (user credential validation, session creation).
     + **Token Service**: Responsible for managing tokens (JWT generation, token expiry, refresh logic).
     + **MFA Service**: If multi-factor authentication is needed, handle it in a dedicated service.
     + **Role Management Service**: Focuses purely on checking and assigning user roles.
     + **Session Management Service**: Manages sessions and tracks user activity.

By **separating** concerns, each module has **higher cohesion**, focusing only on a specific task.

1. **Reduce Coupling by Using Interfaces or API Layers**:
   * Use **Interfaces** or **APIs** between modules. For example:
     + Instead of tightly coupling the **User Authentication Service** with the **User Database**, create an interface (e.g., **IUserRepository**) that the authentication service interacts with. This will make it easier to switch databases or change the implementation without affecting the authentication logic.
     + The **Session Management Service** and **Authorization Service** can interact with each other through well-defined APIs or interfaces, reducing the direct dependency between them.
   * By abstracting the **dependencies**, modules become **loosely coupled** and easier to maintain or replace.
2. **Implement Dependency Injection**:
   * Use **dependency injection** (DI) to inject dependencies like **Password Encryption Service**, **User Database Service**, and **Session Management Service** into the **User Authentication Service** at runtime. This will decouple components and make the system more flexible and testable.
3. **Create a Modular Authentication Layer**:
   * Implement a **modular authentication layer** that allows easy additions like OAuth, SSO, or social logins without changing core authentication logic. Each new method (OAuth, Google login, etc.) can be plugged in as a separate module.
4. **Use of Event-driven Architecture**:
   * Use an **event-driven** approach to handle changes across different modules. For example, when a user logs in, an event (e.g., userLoggedIn) can be published, and other modules (e.g., **Session Management**, **Authorization Service**) can subscribe to this event to perform related actions. This reduces direct dependencies between services.

**Connection Between Components After Modifications:**

1. **Login Service** (Cohesive):
   * Connects with **IUserRepository** (interface to database) and **Password Encryption Service**.
   * Uses the **Token Service** to generate tokens after a successful login.
2. **Token Service** (Cohesive):
   * Handles token generation and validation, decoupling token logic from the **User Authentication Service**.
   * Communicates with **Session Management Service** to handle session-based token expiration.
3. **Session Management Service** (Cohesive):
   * Manages user sessions and session tokens, uses the **Token Service** for managing JWTs and session expiration.
4. **Authorization Service** (Cohesive):
   * Handles access control based on user roles and permissions, uses interfaces to interact with the **Role Management Service**.
5. **Password Encryption Service** (Cohesive):
   * Only responsible for password hashing and encryption, isolated from other services, ensuring a focused responsibility.
6. **Interfaces and Event-driven Communication**:
   * All services are connected through **well-defined interfaces** (e.g., **IUserRepository**, **ITokenService**), which reduces **coupling** and allows flexibility.
   * **Event-driven architecture** allows asynchronous communication between services, further decoupling the components.

**Conclusion:**

By applying these modifications, **coupling** is reduced, and **cohesion** is improved across the system. Each module has a clear, focused responsibility, and dependencies are minimized. The modular design enhances scalability, maintainability, and flexibility for future feature additions (e.g., MFA, OAuth) while maintaining a clean and maintainable architecture.