**CS 230 Module Four Journal: Client-Server Architecture Reflection**

**Client-Server Pattern**

The client-server pattern is a foundational architectural approach that separates concerns between a central server more clients. This model enables applications like The Gaming Room "Draw It or Lose It" to be modular, scalable, and platform independent. by isolating logic between the client and server, developers can implement updates, features, and optimizations on both side without disrupting the entire system.   
For a web based game application, this pattern ensures that clients running on different operating systems like Windows, macOS, Android can communicate with a server through standardized HTTP methods. RESTful APIs act as the communication bridge between both ends, allowing for flexibility and compatibility across multiple environments.

**Server Side**

In this module, the server-side component was developed using Dropwizard and a REST API secured by Basic Authentication. The server exposes endpoints such as GET and POST through annotated controllers, allowing for interaction with user data stored in memory or a database. By using annotations like @Path, @GET, @POST, and @RolesAllowed, the server defines clear rules for data access and authorization, ensuring only authenticated users can access or modify specific resources.  
This RESTful design supports stateless interactions and allows the server to scale independently from the clients. The server acts as the central authority that handles validation, business logic, and security before sending back structured JSON responses that any platform can parse.

**Client Side**

On the client side, the game application must be built in a way that supports communication with the RESTful API provided by the server. To support multiple environments such as web browsers, Android, and iOS, developers should implement HTTP clients using libraries appropriate to each platform. These clients should be able to handle authentication headers, send JSON payloads, and parse server responses.  
To add more users to the system, developers could expand the backend to support a database like MySQL or MongoDB and implement user registration endpoints. Client interfaces would need to support secure sign-up and login forms, possibly incorporating for improved security.  
Future enhancements could include integrating game leaderboards, real-time multiplayer capabilities, chat features, and push notifications for game events. These enhancements would likely require WebSockets or long-polling for real-time updates.  
If the Gaming Room chose to host the app on XBox and PS4, the core REST API would remain unchanged. Developers would need to build native clients for each platform that conform to console SDK requirements while maintaining proper authentication and session management with the server.

**Conclusion**

The client server supported by a REST API allows flexible and scalable development across many platforms. With a secure, centralized server and adaptable client interfaces, The Gaming Room’s application can grow to support new features, users, and deployment platforms with minimal friction. This module highlights how architectural patterns and secure communication protocols can be used to meet real-world software requirements efficiently.