**Module 4 Journal**

**Client-Server Pattern**

The client-server pattern splits the responsibilities between two sides. The client (a browser, mobile app, or console) sends requests, and the server processes those requests and sends back a response. This makes it possible for the same game to run on many different platforms without rebuilding everything from scratch.

For the game app, this means the backend can stay the same while clients like Android, web, or even new systems in the future simply connect through the REST API. This separation of concerns makes solving problems easier and helps the application grow across platforms (Spacey, 2016).

**Server Side**

On the server side, I built a REST API with Dropwizard. REST keeps communication simple by using HTTP methods like GET or POST and returning JSON data. For example, if a client requests a list of users, the server responds with that list in JSON format.

Security comes from authentication and authorization. The Authenticator checks if the username and password are correct, and the Authorizer checks what role the user has (like USER, PLAYER, or ADMIN). The principal object represents the logged in user. With annotations like **@PermitAll** or **@RolesAllowed**("**ADMIN**"), the server can control who has access to different parts of the application. This setup keeps communication smooth while protecting the system (Fielding, 2000).

**Client Side**

On the client side, the main job is to send requests to the server and handle the JSON responses. Since REST is language independent, the same API can be used across web, Android, or another client as long as developers format the requests correctly.

**Next steps for the game app could include:**

**Adding more users:** Expanding the database so players can make their own accounts instead of just using built-in ones like guest, user, player, or admin.

**Adding features:** Things like leaderboards, multiplayer lobbies, or new roles with different permissions.

**Adding clients:** If the app expanded to Xbox or PS4, developers would only need to build the client software for those systems. The backend server wouldn’t change the REST API would still handle the communication (Gupta, 2025).

Working with the client-server pattern and REST showed me how keeping the server and client responsibilities separate makes everything easier to manage. The server handles logic, data, and security, while the client just sends requests and displays results. This separation makes the system more flexible, scalable, and secure.

**References**

Fielding, R. (2000). Representational state transfer (REST). Doctoral dissertation. University of California, Irvine. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\_arch\_style.htm

Gupta, L. (2025, April 1). What is REST? REST API Tutorial. Retrieved https://restfulapi.net/

Spacey, J. (2016, December 27). What is separation of concern? Simplicable. https://simplicable.com/new/separation-of-concerns