Joven Rosales

CS 230

3-30-2025

CS 230 Module Four Journal

Client-Server Pattern in Software Development

The client-server pattern is a fundamental approach in software development that helps separate application logic into distinct parts. This pattern is crucial for developing applications that run on multiple operating platforms, like our web-based game application. According to Sommerville (2015), "The client-server model is a distributed system structure that splits functionalities between service providers (servers) and requesters (clients)." This separation allows flexibility, ensuring that the client side can focus on user interactions while the server side handles data processing and business logic. By using this pattern, our game application ensures a seamless experience across different platforms while maintaining efficiency and scalability.

On the server side, the application is designed to handle requests from multiple clients through a REST API. Representational State Transfer (REST) is a widely used architecture style for web services. Fielding (2000) described REST as "an architectural style that emphasizes a uniform interface and stateless communication between client and server." This principle allows our game server to manage authentication, authorization, and data retrieval effectively. We implement authentication and authorization mechanisms using the principal object, an authenticator, and an authorizer, ensuring that only authorized users can access game-related data. The use of annotations like @PermitAll and @RolesAllowed("ADMIN") helps enforce security by restricting access based on user roles.

On the client side, developers must ensure that the application is accessible across different environments. The game currently supports three clients, requiring developers to follow best practices to maintain consistency across platforms. According to Mozilla Developer Network (n.d.), "A REST API enables applications to communicate using standard HTTP methods, making them adaptable to different environments." This adaptability ensures that our game can be used on various web browsers and mobile devices without extensive modifications. To add more users to the database, developers can implement registration and login features that securely store user credentials. Additional features like multiplayer capabilities, leaderboards, and in-game purchases can enhance user engagement.

Expanding the game to additional clients, such as Xbox and PS4, would require further development considerations. These platforms have specific software development kits (SDKs) and authentication requirements that must be integrated into the existing system. The REST API would still facilitate communication between the client and server, but developers would need to optimize the game for each platform’s unique hardware and input methods. By following REST principles and maintaining a modular architecture, the application can be extended without major restructuring.

In conclusion, the client-server pattern provides a robust framework for developing scalable applications across multiple platforms. The server-side implementation ensures secure communication and data management using REST API principles, while the client-side development focuses on user experience and compatibility. As Fielding (2000) emphasized, "A well-designed REST architecture supports scalability and flexibility in network-based software systems." By leveraging these principles, our web-based game application can continue to evolve and support additional clients while maintaining security and performance.

References:

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

Mozilla Developer Network. (n.d.). REST APIs - MDN Web Docs. Retrieved from https://developer.mozilla.org/en-US/docs/Glossary/REST

Sommerville, I. (2015). Software engineering (10th ed.). Pearson.