Chris Williford

Southern New Hampshire University

CS230: Module Four Assignment: Journal: Software Application Requirements

Tracey Lanham

March 27, 2025

The client-server architectural pattern is fundamental in developing applications that require interaction between different components. In the context of a web-based game application, this pattern allows for the separation of concerns, where the client side handles user interactions, and the server side manages data processing and storage. This separation enables the application to run on multiple operating platforms, such as Windows, macOS, and Linux, as well as mobile platforms like iOS and Android.

By utilizing the client-server pattern, we can ensure that the game application meets software requirements efficiently. For instance, the client can be developed using web technologies (HTML, CSS, JavaScript) that are universally supported across different browsers, while the server can be built using a robust backend framework like (Node.js, Django, or Ruby on Rails) that handles game logic, user authentication, and data management.

The server side of the application plays a crucial role in facilitating communication with the client side through a REST API, REST (Representational State Transfer) is an architectural style that uses standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources.

* **Communication:** The server exposes endpoints that the client can call to retrieve or manipulate data. For example, a GET request to `/api/games` could return a list of available games, while a POST request to `api/users` could create a new user in the database.
* **Data Handling:** The server processes incoming requests, interacts with the database to fetch or update data, and sends back responses in a structured format (usually JSON). This allows the client to easily parse and display the information.
* **Scalability:** By using REST APIs, the server can handle multiple client requests simultaneously, making it scalable and efficient for a web-based game application.

For the client side, developers must ensure that the application is compatible across various environments (web browsers, mobile devices, etc.). Some considerations include:

* **Responsive Design:** The application should be designed to adapt to different screen sizes and resolutions, ensuring a seamless user experience on desktops, tablets, and smartphones.
* **Cross-Platform Compatibility:** Developers should use frameworks like React, Angular, or Vue.js that facilitate building applications that work across different platforms.
* **API Integration:** The client must be able to make API calls to the server to fetch game data, submit user actions, and receive updates in real-time.

**Next Steps for Client-Side Development:**

* **Adding More Users:** To add more users to the database, the client application should provide a registration form that collects user information (username, password, email). Upon submission, a POST request can be sent to the server’s `api/users` endpoint to create a new user.
* **Additional Features:** Potential features to include in the game app could be:
* Leaderboards to track player scores.
* In-game chat functionality for player interaction.
* Achievements and rewards system to enhance user engagement.
* **Hosting on Additional Clients:** If the Gaming Room requests to host application on Xbox and PS4, developers would need to:
* Adapt the game to meet the specific requirements and SDKs of these platforms.
* Ensure that the game can communicate with the server using the same REST API, possibly adjusting for platform-specific features like (controller input).
* Test the application thoroughly on these platforms to ensure performance and user experience are consistent with other clients.

By following these steps and considerations, the web-based game application can be effectively developed and maintained across multiple platforms, ensuring a robust and enjoyable experience for all users.