The core fundamentals behind client-server architecture are a computer network framework in which remote processors transfer information from a centralized server. Client computers provide the user with an interface to request the services of the host server; the product of the output is the coordination of the server giving the information to the client. In other words, servers and clients “talk” back and forth, much like a conversation between two people. Typically, a server provides a standard interface to clients so that the client would not need to know the specifications of the system they are retrieving the information from. An example of a client would be a desktop computer set up at an office or laptop, and even mobile devices are clients. Servers are located remotely and are powerful computers capable of handling the requests of multiple clients. Some servers are not just one computer but multiple computers clustered in tandem. Servers like these are used for cloud hosting or large social media platforms like Facebook. This computing model is highly effective when clients and servers have designated methods that are performed routinely. Clients can retrieve information from the server in high volumes if necessary, while the client can perform local tasks. This is how a web/game-based application could run on various operating platforms.

Application state and functionality are distributed into resources, and the application responds based on the resource of the operation being performed. Additionally, resources are addressed using URLs that can be utilized as external links (for instance, <http://server.com/resources>). For each corresponding resource, the data type of that resource will be added to the URL (<http://server.som/resources/resource>). Resources are a means to describe the information using Multiple Purpose Internet Mail Extensions (MIME) types supported by HTTP. Although REST is specifically designed for web applications, its practicality is entirely modular. For instance, the client and server are separate entities and source code on the client side can be modified without messing up the server end’s functions. The most basic functions of the REST interface are HTTP verbs such as GET, POST, PUT, and DELETE. A client will input these keywords into the header of the request to the server, the client will select the content that will be distributed from the server. The space from this request is called the “accept” field that regulates the client's input so that the server does not send unnecessary data that may not apply to the request.

Protocols are defined as stateless, cacheable, and layered. Stateless protocols have little to no issues because connections are nonexistent in UDP, multiple users could send these packets through the same port, and the packets will be delivered in a random sequence. Unlike stateful protocols, a connection is identified by a 4-tuple map, the instructions embedded within a protocol like TCP will contain information such as the source and destination ports, as well as the IP addresses associated with them. This is beneficial in the case where two different machines are connected to the same port, and the connections will be different because the IP addresses are not matching. In the case where the machine connects twice, the connections are identified by the source port. In layman’s terms, if a user was to connect to the same server twice from the client, both connections will have different source ports from the interface and the destination ports of the server. Ports are essential to IP addressing because it is a multitudinous way of identifying each connection, different applications running the same protocol, and IP can listen on the ports. In some cases, an application may define a higher-level protocol, so a multiplex port might not work. If two connections simultaneously use the same TCP protocol where the source and destination IPs and ports are the same, then the signal would be a shared connection.

Given the current situation where the gaming room must be hosted through to the client, the DevOps must consist of the following practice: frequent stability checks for connectivity, securing the streaming sessions through encryption, maintaining user authentication in the gaming room, and minimizing the load on the client side, so hardware specifications do not take up expenses.

Proper load balance, rendering, and graphics should be carried out on the server side, and video streaming is dependent on the client side. This makes sure that an encrypted stream responds in accordance with the input from the client. This is also a way to ensure that there isn’t a load on the client end to process instruction considering this is the role of the server side. A back-end system with access to the database might provide methods for user management, for instance, a user entering into the session or accepting multiple inputs from users. This could open up multiple clients being connected to a single server. There should be an emphasis on proper load balancing, and the server should at least be able to handle 10 users simultaneously, and there should be a capacity on the number of users on the server at once since it would be less stable if there were more users there should be implementations to the limits of the number of users.

One way to add users to the service, would be to expand the database for the game. In addition, it would be necessary to develop server-side support for more users accessing the system. Once you expand the server's capabilities, you have to have support in terms of software and hardware to ensure that the user is having the best experience possible. Also, consider networking and the latency between the client and the players. The better networking tools that are available to maintain and continue development on this game, the more users will play because of how reliable hosting is. Rewriting code on demand is a benefit of using a server-client-based architecture. It should also be mentioned that expanding this game to PS4 and Xbox players will require further development, however maintainability of these consoles are limited to how long they stay supported by their respected companies. The consoles mentioned have capabilities of streaming TV and video games directly to their system. If the consoles were to be added, it would be like plug and play. Development efforts on the console end will be finished, and the server-side will be able to make it’s implementations based on the client’s development notes.