Therefore, when understanding how a client-server pattern/model works, we need to comprehend the basic components that form its foundation. To understand, we need to break it down into its two main parts of client and server. On the client’s end, the term “client” refers to the device used, which could be a device like a computer or smartphone, or an application such as Yahoo or Safari. On the server side of everything, there is a system that provides the data and resources needed to process any incoming request from the client side. So, in conclusion, the client sends a request to the server, then the server fulfills the request, which then in turn sends a response back to the client side. This is what makes the computer systems work. On top of this, it also allows for data management and resource allocation, which in turn has a handle on large-scale data storage and complex processes. This also allows for scalability and resource optimization. So, for our web-based game application, this can fix problems with intake, resource allocation, and when multiple clients are trying to access a server, it allows for reliability. On top of all these pros, it can also give a sense of security and stability, which gives privacy to any user.

Now that we have discussed the basics of the client-server pattern, we can go to greater depth in each part. So, to describe the client side in more detail, we need to start by understanding REST API style. According to resfulapi.net, REST or “Representation State Transfer and architectural style for distributed hypermedia systems” (Gupta, 2025) is one of the most “widely used approaches for building web-based APIs”.(Gupta, 2025) REST has six principles: uniform Interface, client-server, stateless, cacheable, layered systems, and code on demand. So, to first describe a uniform interface, it means that it can be simplified and improve the visibility of interaction. Client-server enforces separation, which helps the components of the server evolve on their own. Stateless means that all information between the server and client is easily understandable and also contains all the information. Next is cacheable, meaning that the client application gets to “reuse the response data later for equivalent request” (Gupta, 2025). Next up is the Layered system, which is composed of layers in a hierarchy. Lastly is the optional code on demand, which simplifies the downloaded code. This quick and basic overview gives a similar look into the REST style. REST often uses Uniform Resource Identifiers or URI to help HTTP methods become more uniform, which helps access other content like JPEGs and HTML links. Along with uniformity, REST is also a resource that consists of interlinked resources. This is often shown through hypertexts, which in turn allow the user to easily navigate from one target to another. This is what helps the server side provide communication to the client side through the REST API.

On the client side, it is the developer’s job to make sure that the application is flexible to have the ability to grow as more clients come on board. Another rule, according to geeksforgeeks.org, is centralized management. This allows moderators to “manage data, apply updates, and enforce security policies.” Just like the server side, the client side also has steps for its design. The first design is requesting a website, this is followed by the content delivery network, which gives the content to the client. Next up are the downloads of HTML and JavaScript that are connected to it. Then there are the execution and API, followed by the server response with the proper data, and finial the data fills the placeholders. So, to add more users to the database since the server side is flexible, the client could be added through a separate and unique identification that would be linked to their profile. Some other features I would add to the game app are the ability to run on both platforms to allow both PC’s and phones to run the application. This can work for cross-play, and to host more clients, the data for the game would need to be simple enough to implement into the varying systems. Each system would need to be labeled uniquely, with it having its line of code to perform properly within its system. For instance, an Xbox is not going to run the same way a PlayStation or phone would. The developers would need to adapt the code based on the system. Then each system would have its line of code that could be accessed through the unique ID the player was given, so only Xbox can access the Xbox code, PlayStation the PlayStation code, and so on and so forth. This would allow for the Gaming Room to add as many clients as they wanted while still being able to accommodate previous clients.

**Citations**

N, Navlaniwesr, & Follow. (2025, July 23). *Client-server architecture - system design*. GeeksforGeeks. https://www.geeksforgeeks.org/system-design/client-server-architecture-system-design/

What is rest?: Rest api tutorial. (n.d.). https://restfulapi.net/