The client-server pattern can be used to satisfy software requirements and efficiently solve problems. This form of structural pattern is one of the most common patterns to use. This pattern allows for a centralized system with all the data in one location, language communications, and server and client scaling over time. This can be simplified by the server being the host or central hub and the client being the end-user. The client interacts or requests data from the server, which is easily accomplished by having a centralized system with all the data in one place. This concept also allows for resource distribution since the client doesn’t share resources with the server. The client only sends requests to the server for data, which is then processed and returned. The separate and distinct role of a client interacting with a server enforces the well-established design principle of separation of concerns: a consumer (client) of remote resources that are maintained by a producer (server). The most predominant example of this is accessing a webpage on the internet. By entering a URL, a browser (client) can access all forms of data types from the requested webpage (server). This is normally accomplished using the HTTP protocol system.

RESTful API is a common term in the technology community today. REST stands for “REpresentational State Transfer.” When the client requests to access information from the server, it responds in a REST API format which uses the distinct URL to determine which data is being requested and responds with an HTTP request. This is normally processed in JSON format, due primarily to the fact that it can be easily read and written by people and easily parsed by programming languages. JSON is a text syntax that facilitates structured data interchange between all programming languages. This enables the developer to protect data behind security measures, such as username and password systems, as well as allows for layering which promotes performance.

There are several guiding principles of the REST architecture: uniform interface, client-server, stateless, cacheable, layered system, and sometimes code-on-demand. A uniform interface is accomplished by abiding by four constraints: Identification of resources unique to the client and the server, manipulation of resources through representations in server response, self-descriptive messages that carry enough information to describe how to process the message, and hypermedia as the engine of application state which allows for the client to drive all resources and interactions by use of hyperlinks. Client-Server enforces the separation of concerns, which helps the client and server to evolve independently. This is important because by creating a divide between the client and the server, we improve the portability of the user interface across multiple platforms. Stateless requires that each client request to the server must contain all the information required to understand and complete the request correctly. Cacheable constraints are important because they determine whether the client request can be reused by the server later. The layered system is important to improve performance by only allowing the instance to interact with the layer that they are currently on. Code on demand, while optional, is good for optimization because the server is not directly processing the request from the client but is sending executable code to the client for its execution. This may not always be desired based on the client’s specifications and ability to process that code.

All client requests should be able to communicate with the server. By utilizing an encrypted stream session, more users could be added to the database. A server would respond to different clients in that server instance. Load balancing is also necessary to not hinder processing client-side. If all graphics, rendering, and other heavy processing tasks can be accomplished server-side, then this will free up client resources. The demands of the server should also be capped to a particular number of users to not slow down the server too much. All further requests from clients to join the server should be maintained in a queue. This is common among current video games with large player bases, like Destiny 2 and World of Warcraft. By utilizing these features and development schemes, integrating Xbox and PS4 compatibility would be relatively easy.

*Dropwizard Client — Dropwizard*. (n.d.). https://www.dropwizard.io/en/stable/manual/client.html

Gupta, L. (2022). REST Architectural Constraints. *REST API Tutorial*. https://restfulapi.net/rest-architectural-constraints/

Lavieri, D. E. (2019). *Hands-On Design Patterns with Java*. Packt Publishing. https://mbsdirect.vitalsource.come/books/9781789809954