The client-server architectural pattern is effective and extremely scalable. Data is stored and manipulated on the server, and the client is only given access to the user interface. For this reason, the client-server pattern is effective in meeting software requirements, as the client (the player of the game) does not need to know where and how their actions in the game are being logged, they just need access to play. This allows for the game to be accessible on different platforms, as the client-side portion is modifiable to meet different platform requirements while the server maintains its functionality unchanging.

The server’s form of communication is a response to the client’s request. It does this through the usage of the REST API. Typically, the REST API will make use of HTTP methods (GET, PUT, POST, DELETE) and URLs in order to request information from the server. The server will then process that request and return the appropriate data in the form of JSON, while ensuring that communication remains stateless. JSON is JavaScript Object Notation, and is easy to define and parse, making it perfect for a response. Keeping the communication stateless means that each request and response from the client and server should be separate and self-contained. The server does not need to know the state of the client for a request, which means that the client should include all appropriate information in order to get the correct response from the server.

To have the application on three (or more) different clients work on the site, the developers have to ensure the client-side framework is able to properly communicate with the server and have a workable user interface. The server, which was written in Java, should remain independent of the differing client’s circumstances. This should allow the developers to create the client side specific to the platforms. In a web browser, the communication would likely come in HTTP methods, while the user interface is displayed with HTML, CSS, and JavaScript. Expanding to the world of PlayStation or Xbox would more than likely require the client-side (i.e., the game) to be developed in C++, and having a mobile app version would likely require it to be built in a language like Java (for Android) or Swift (for IOS). The point is that client-side development is likely going to be different on different platforms, which can be seen with popular apps like YouTube, Spotify, and Netflix, which can all be accessed through their website, their mobile apps, or their game console apps. They all access the same data through their server, it is just being served differently client-side. The next most important point for the developers to consider is adding new users. Currently, the demo app has three hard-coded users with different permissions, a guest, a user, and an admin. When the game is accessed on any platform, the client should be prompted to register or login, and the information they provided should be passed to the server, where it is then checked with the database, and then the appropriate response should be returned to the client (i.e. “Registration Success” or “User already exists”). An additional feature to consider would be live chat, as a lot of party-type games are played online, and having the extra form of client to client communication would likely be considered a plus by most users.