Cameron Thorp

May 28, 2023

Module Four Journal

**Client-Side Server Pattern:**

Using this design pattern is ideal for an application that requires users to have access to the same instance of an object. An example of this would be a game that needs to be running in the same state for all players connected remotely. This pattern also allows for the developer to offload some of the processing requirements to the client computer.

This client side processing means that users have faster access to data unique to them, as it can be calculated locally and then sent to the server in the finalized format to sync with other users. In addition, it allows the synchronization process to occur faster since the server is primarily responsible for only managing HTTP requests and possibly performing operations involving information from multiple clients.

**Server Side:**

Using REST API style for the server side allows for a common communication protocol. This commonality encompasses the commands used to communicate between the clients and the server, and the format that data is passed between the clients and the server.

More specifically, using REST API styles allow the functions on the client side to utilize HTTP protocol to communicate with a server easily. These include GET, POST, PUT, DELETE and more; each of which requests different actions to be performed on the server side. They can also have errors explicitly defined to help gracefully handle failed actions.

The way these commands transfer data to the server side is via common formatting. This includes XML, JSON and YML as the most common formats. Passing data in these formats is supported by a majority of languages and frameworks, allowing for data retention and error reduction across Operating Systems, applications, and architectures.

**Client Side:**

When developers create an application where clients act as users on the website, they must account for extra functionality depending on the client. This is managed by creating roles that can be set and retained through the use of a database, or by being hard-coded into the server. These roles will often-times be assigned via username and password that is entered by the client, giving them access to the functionality granted to that role. This functionality is accessed by utilizing REST APIs combined with function role assignments to ensure only users who have been authenticated in the database have access to functionality which could impact the server or company.

In order to add more users to the database, it would be possible to create functionality to do one of two things. The functionality could be to prompt entry by a client for new account information, which would then be translated into a common data format (likely JSON), and then that object would be sent to the server to be verified and potentially added to the current user JSON file. The other approach would be to create a server-side interface with a database that would keep track of all user information. The benefit of the database would be that, once the data is successfully added, it would be immediately available to GET methods (meaning the user could log in automatically without needing to wait for the JSON file to be updated).

The first feature I would add to the game app would be to create a user database so that users could more easily be created. Along with this, I would likely modify the admin functionality so that they could manage users more efficiently as well.

If the application was to be hosted on a client such as Xbox or Playstation, the most difficult aspect would likely be creating a client-side GUI as their interfaces are different from a standard web-browser. It is likely however, that the standard HTTP protocol would still be accessible as long as the console is connected to the internet, so once the client-side interface is set up, they should have access to the same database and states that a standard web-browser would have.