Daniel Leon

CS 230 Operating Platforms

Southern New Hampshire University

4-6-2024

**Module 4 Journal**

The client-server pattern is basically a service being requested and then provided. The client, maybe your computer, requests information or a service via a network such as the internet, and the server, either a centralized hardware computer or a cloud server, sends back the service that was requested. The server side is able to handle requests from multiple clients at a time operating on multiple different platforms. Most of the processing that takes place during these requests is handled on the server side, which takes a lot of the workload off of the client side. This is beneficial as it enables the client side to operate on a wider variety of platforms without having to carry any extra load. Another benefit is that all data files are saved and managed in a centralized location.

We communicate from server side to client side using the RESTful architectural style. There are six guiding principles of this style of communication. The first is to have a uniform interface, basically a way to communicate between multiple different platforms using multiple different languages. This can get confusing and cause problems, so most RESTful APIs use a common language between client and server called HTTP. The client request contains a method (either GET, PUT, POST, or DELETE), and a URL, which is the endpoint where the server side meets the request. The server side then sends back what was requested in JSON format.

The second principle, or constraint, of RESTful is the client-server pattern which calls for a separation between the two. This separation says that only a client can request and only a server can respond. This helps both to evolve independently because any changes made to either side in their respective evolution does not affect the other side. The third principle is statelessness. This is where every single interaction, or transaction, between client and server has every piece of information needed to complete this exchange, and each time is treated as a brand-new request with no knowledge of past requests. This reduces the amount of memory that is used by the server and greatly increases the amount of successful exchanges. Fourth is a layered system, which says that no matter how many layers between client and server, the exchange must stay the same as mentioned above. A request might go directly to the server it is intended to reach, and another might have to go through another server, maybe a security layer, to reach the intended server. No matter how many layers, the exchange remains the same.

The fifth principle would be cacheable, where caching is storing data on the client side for future interactions with a particular site. What this does is helps speed up web page load time for the client and lessens the workload for the server. When a resource is requested, the server sends it in a response. Caching must take place if it applies to said resource and said resource must announce itself as cacheable. The last principle is actually optional. It is code on demand, which is basically the option for the client to request code to be sent back in the server response.

For the developers of Draw It or Lose It, what was required was to make the game available across multiple platforms. One of the best ways to do this is to implement the RESTful API architecture, which by its nature needs to keep security at the forefront. The way to do this is by incorporating authentication and authorization into the program. The Authenticator class basically checks for matches to usernames and passwords to authenticate the user. Once that is completed, the program then checks the user’s level of authorization. The Authorizer class defines what it means to authorize a user by using a getroles() method. In the GameUserRESTController class we use annotations such as @Auth and @RolesAllowed to allow each user specific permissions. All of this works together for a more secure experience.

As for the current users of the game, I would make “admin” only accessible to our client, The Gaming Room. To add new users to the database, I would link our addplayer() method from project one to our Authenticator and Authorizer classes to secure each user and apply roles as necessary. If I were asked to host the game on PS4 and Xbox I wouldn’t have a problem with it. I think that the RESTful style would transfer over with little to no issues. I would of course have to implement some changes on the back end as far as coding goes, but I see that as another challenge that I am more than willing to meet.