Zachary Meisner

1/29/22

Operating Platforms

Prof. Fredericks

- **Client-Server Pattern: Discuss how the client-server pattern can be used to satisfy software requirements and efficiently solve a problem**. Specifically, the web-based game application must be able to be run on multiple operating platforms.

The Client-Server Pattern is a valuable tool to use when creating a web-based game application. Creating an environment where different operating platforms can integrate within the scope of the project is pertinent to not only security, but also ease of use for both the players of the game, including efficiency for those who run and moderate the game as well. To explore some of the potential software requirements we may have, it is important to think as if we are developers, and to think as developers, we must think as the target audience. The problem of this in total is enigmatic and has proven to be one of the most challenging tasks for companies that create and run software applications. To put it simply, we must put some thought into the architecture of the environment by considering the differential within the audience. This will help enlighten the ideological structure that we are able to create, as we can account for all possibilities, in addition to preparing for the worst-case scenario, such as a data breach, or a server crash. The most valuable part of this pattern is the ability to configure and fine tune the application to morph the product in any way we need. In addition to this, we are also able to account for the configurability of the server, so it may respond to any requests in the appropriate manner. For example, knowing that the end state of the application during runtime must remain constant, we can then deduce the production of the server using basic logic. If our application runs well with windows operating systems, we can see the requests from the users as they request to visit and connect with the host, in addition to seeing what else they are able to request and watch how the application acts in runtime before we even configure the ability to allow other operating systems to connect as well. In this way we can find any security holes, or breaches of our initial logic within the program, that may misconfigure the structure. This incremental testing is crucial, as quality assurance must be there to deliver the final product accurately. The allowance given by this quality assurance helps us gauge culmination within production, so we may define the point in which we are able to introduce connectivity between different operating systems. What is most important at this point is that if we were to consider the full stack of a program visually, a popular chart to use is of that as a hamburger. The relativity in this example shows us that when choosing what to put on your burger, or even what kind of bread to use, it changes the taste of the final product; and given that the most important part of creating a hamburger is when you eat it, we must use the finest of ingredients.

- **Server Side**: You have developed the application from the server side. **Discuss how the server side provides communication to the client side with REST API style**.

As previously discussed, when integrating the ability to allow the Client-Server pattern to allow different operating systems to connect to your web application with ease, incremental programming is crucial for the final product so we can assure the quality of our programming to prevent any bugs in the logic of our code. What is important to consider here is that every operating system in some way is a reiteration of itself, but on a much larger scale. The mathematical positioning using binary calculus to decide the state of being in which a computer ends is outside of the scope for this article, though we can draw relativity using that of a rainbow. The spectrum of light emitted from a rainbow projects a differential of colors within the stacked ideological state of being of the rainbow itself. What is most interesting about a rainbow is that it is an illusion, dictated by the positioning of where you are standing and looking from the ground up into the sky. REST API style architecture between multiple operating platforms works in an incredibly analogous way. If the operating system trying to connect to the projected server is not within the accounted scope of the system at play, the operating system will not be able to efficiently connect and use the application. In other words, it is not standing in the correct position to see the rainbow. Now, if the operating system is within the accounted scope, then the successful occurrence of TCP/IP is utilized to communicate between the user and the server. REST stands for Representational State Transfer which uses the logic of the TCP/IP connection to return the state of being of the program at the given state of time, in addition to accounting for the allotted instructions encoded within the API. In other words, if you are in the correct place where you can see the rainbow, what colors would you see? Each band and color of the rainbow can be thought of as the emitting rays of sun at the given point of time; based off the initial parameters that dictate the content that you comprehend when you reflect on the sight of the rainbow.

- **Client Side**: You wrote an application for multiple clients where the multiple environments can interact with the server. **Discuss what is required of the developers so that the application on all three clients is able to be used on the website**. Consider what next steps would entail to develop for the client side of the game application. For instance:
    - How would you add more users to the database?
    - What other features might you include in the game app?

- What if The Gaming Room asked you to host the application on a fourth and fifth client? For example, on Xbox and PS4.

Apart from planning the design and architecture of the application, the summation of requirements boils down to the state of being in which the application itself is being presented on the internet. The state of the API is most important, as the instructions for the application are what provide accountability for the constant state of being required for each of the different operating platforms, in addition to adding new ones such as Xbox and PS4. Automation here is just as important, as we can account for the instances of each request based off the operating platform so they may connect, which we are also able to use to add more users to the database. These requests from the users show the need and demand from the encapsulated program and allow passage in and out from the metaphorical castle we have built to promote singularity for all users to protect their information. This is done by integrating the base logic of the application within a user interface that allows them to register themselves as a user and set a password. The communication between the user and the database are foundational steps needed to create scalable interoperability for the application. Creation of the application architecture begins to approach zenith as differential clarity starts to peak through due to the established a point of singularity at a base level; allowing us to gauge the need for distinctive features. For example, during runtime if we have found that users may benefit from an interactive AI chatbot, not only are we able to improve the underlying systems to help support the structure and integration of the application on the different operating systems, but we are able to manipulate the application at the final layer of the full stack to act as a constant with the other existing features in the application.