Daniel Loranger

CS-230 Week 4

Journal Submission *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.*   
  
Using the Client-server pattern for the gaming room is ideal as its widely used in internet services and is widely adaptable.  Having a centralized server, which behaves as a provider device that answers requests from client devices, allows a single point of control for all the shared resources needed to support widely distributed users. This ensures all users are operating on the same software base which helps to ensure consistent behavior as well as allowing for near real time deployment of software patches, and expanded offerings for a better user experience.  
  
The client-side devices are considered to be a requester type device that can be tailored to the user’s unique hardware such as mobile phone, game stations, desktop computers, etc. This allows the client software to be implemented in different programming languages, display resolution nuances, etc. as may be required. Additionally, so long as the data communication formatting to/from the server doesn't require changes, the individual elements can be developed concurrently with minimal impacts to other teams as long as shared code is managed properly to ensure compatibility.    
  
One key constraint when utilizing the server in a remote client network is efficient data transfer as the server needs to handle traffic to a very large client base under varying load conditions.  
  
An additional key constraint for a centralized server, reliability is of utmost concern as it is a single-point of failure that can break the workflow for all remote clients.  
  
A familiar example of the Client-Server relationship is the internet which uses servers to host websites and clients which request data and then render locally on the client devices.

All of these details directly contribute to satisfying software requirements by establishing a well defined implementation of what the server is responsible for and what the client is responsible for, along with the connectivity and data flow between the two. Further, by breaking up the software into well defined buckets with a well defined communication interface, automated testing will be easier to implement as a distinct test client/server/sniffer entity that can be directly coded to perform regression testing which helps identify coding gaps, regressions, etc., thus ensuring requirements are maintained, and well behaving updates are published.

*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.*    
  
The server hosts the REST compatible API for the application. This API is accessible typically via a network connection, but could be by other means as well. Through the use of a well-designed API, the server does not keep track of any specific state of information, thus allowing the client to request any information at any time as necessary to complete the needed task.

For this demo example, the client is a web browser that requests to log-into the server via some pre-existing user/password associations. After logging into the server, a different API call checks the user’s authorization status, and then displays a list of acknowledged users, presumably thru a client request that retrieves all users.

The principal object would be the context the code in running under in terms of system security. For example, the code could be running as a read only context thus not allowing any changes to be made to the file system. A different user could be operating as an admin, where the read-only provisions are disabled allowing direct system management.

The authenticator is the mechanism that forces user identification to be established before allowing any access to the REST API resources ([Merge.dev](https://www.merge.dev/blog/rest-api-authentication" \l ":~:text=implement%20API%20authentication.-,What%20is%20REST%20API%20authentication%3F,to%20your%20REST%20API's%20resources.))

The authorizer is responsible for checking the established users access control and ensuring they only are granted access to the resources they are allowed.

Annotations are embedded instructions that guide both development and usage of the rest API ([knowl.ai](https://knowl.ai/blog/a-comprehensive-guide-to-annotations-in-rest-api-cltvaw611003yn2diyzn6ds57))

*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.*

To support multiple clients, first the API must make no assumptions about the state of the client, it must return the data in a format that is client/server state agnostic. Any client at any time will retrieve the same information given the requests to the server are identical. The clients must also interact with the server in such a manner as to not tie up the resources of the server unless expressly intended to by design. This allows multiple clients to send requests and receive responses in a timely fashion. The client application must account for the fact that other clients might be requesting information (thus blocking resources) and allow retries and other mechanisms that allow equitable sharing of the server resources.  *For instance:    
How would you add more users to the database?*

To implement a feature for the client to be able to create new users, the server would need to expose a new function via the hosted API that allows creation of new users, presumably with very limited privileges initially, and then more advanced users could then escalate the new users’ credentials via a separate API call.

Perhaps a multitude of new user roles would need to be established beforehand, to allow super users, administrators, trainee, etc. for user class distinctions.

Each client application would then need to properly implement the procedural requirements and data handling to first request the new user be added, and then validate success, possibly including other checkpoints such as testing the users phone number or other traceable data. *What other features might you include in the game app?*

In addition to the existing log-in feature, the newly created ‘new user’ process, all the basic game mechanics features would need to be handled in the client side game app. These would include start new game, create new team, add user to team, search for user, quit game, resume game, take turns, etc. Some of these would require pushing data to the server, where others might be pulling data from the server such as score keeping, etc.

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

Presumably the server being well established with a client agnostic implementation would not require anything to change, so only the respective client applications would need to be coded, tested and deployed. These client apps would be able to largely re-use the existing client code base for many functions as the code would be established and tested against the server and thus represent known-good code points. Even if the coding language has changed, the basic workflow would not change at all.