The objective behind REST-Style API is based on the interaction between the two tiers (server and client-side) supporting multiple platforms, capabilities, and languages. Server- client relationships allow there to be allocated work as clint shares no resources with server. Behind the structure, client and server are separate entities. While the client request resources, the server provides them. For both, they will grow independently. In addition, the communication between the server side and client is that each request from the client to the server must contain all the information needed to run accurately to continue with the request. The server does not store any client context between the requests. When the client makes the request, it sends an HTTP request to the server (e.g., GET, POST, PUT, DELETE), specify the resource, provide metadata, and containing data for POST or PUT requests.   
There are some benefits to this arrangement:  
\* Centralized Control: servers’ central resources and service, which means all in one spot, making it easier to manage, secure the system, and update.   
\* Scalability: both client and server are independent.   
\* Flexibility: shifting focus on user interaction, while servers handle the not so easy management (data)  
There are some drawbacks to this arrangement:  
\* Single Point of Failure: if the server goes down, clients cannot access all services/resources.

\*Network Dependency: client-server performance depends on the speed and reliability (network)

\* Security Risks: it’s the main target for attacks ( i.e phishing) and security requires a solidified plan.

To be successful as dispersing the applications to other clients we will need to shift focus on several key areas:  
\* Responsive Design- ensuring the web design support the application adapts to the diversity of screen size and orientations. (i.e layouts, grids).

\*Cross- Platform Compatibility- adherence to web standards and best practices to ensure compatibility across other devices. Always a lot testing the application of any browser (i.e chrome, Firefox, Edge) and devices (i.e IOS, Android, MacOS, or Windows) to identify any issues and fix any compatibility issues.

\* RESTful API- this is to handle data exchange between the client and server. Need to make sure API is well- documented and follow conventions. Also including authentication mechanisms to managed user sessions across various clients.

Performance Optimization- implemented of caching strategies to store frequently accessed data locally, while reducing the need for repeated server request.

Security- using data encryption between client and server and validating user inputs to ensure the implementation of security vulnerabilities.

\*Scalability- modular architecture to facilitate easy updates and scalability.

In my opinion, other features that I feel is important is the Cross- Platform Support to ensure that every person can access the game application whether they are using PS4/Xbox, IOS, or simple androids. Compatibility to various platforms and ensure it runs smoothly with interruptions. Saving progress (during games) are very important to gamers and other Professional Gamers as well. Cloud saves should be implemented during usage and prompts users to save at certain times during usage such as reaching the beginning of Level Two (i.e “Would you like to save your Progress?”). To reach the highest level of success for any gaming application is reaching the most audience you can promote for any clients.   
Any feature I would like to support to user feedback and analyzing any issues that users are facing while using the application and investigate progressing the experience while addressing suggested improvements. Using analytics to track user behavior, gaming performance, and engagement metrics. Lastly, in order to create a near perfect application is making it accessible to multiple platforms such as Gaming Systems (Xbox. PS), Androids, IOS, or Windows.