**Brandon Hobbs**

**CS-230**

**Jan 28, 2022**

**Module 4 Journal**

During week 4 a REST API was designed to act as an interface between the client browser and the server. This brings quite a few benefits by separating the concerns into a server-client relationship. This server-client relationship allows there to be a distributed workload as the client does not share resources with the server. The client sends request for data to the server that is then processed and returned.

There are a few disadvantages to the server-client relationship:

* Separation allows each side of the to be distinctly attacked: viruses present in the client may be uploaded to the server or server Denial of Service (DOS) attacks may prevent communication back to the client
* Phishing and Man-in-the-Middle attacks allow the information to be captured at some midpoint and the useful pieces, e.g., login credentials, stolen
* Transmission may be compromised with spoofed packets or even modified

Even though there are disadvantages to this arrangement it has been widely adopted because it has allowed:

* For a centralized system with all the data in a single place
* The server and clients to scale or evolve differently
* Language agnostic communications

There are often two separate approaches to this online data communication: SOAP and RESTful. SOAP is an actual protocol maintain by the W3C Consortium and RESTful is more of an architectural framework. SOAP brings a few benefits, built in security and transaction compliance, but can be slower and heavier to implement.

The RESTful API architecture has 6 attributes:

* There are separate client ad servers
* Each request is stateless – meaning each request contains all the necessary information and does not rely on previous transmissions
* Caching is expected to be used for fast response
* Uniform interface to allow predictable API behavior must allow for unique identification of resources and a common vocabulary for manipulating those resources
* Any layers, e.g., caching security, etc., must be transparent to the client and server
* The server may, optionally, return code for the client to run, such as JavaScript

The REST API used in this assignment was designed to authenticate and authorize certain behavior to differing accounts. This implementation follows the pattern shown in here: <https://howtodoinjava.com/dropwizard/dropwizard-basic-auth-security-example/>

The principal object in security represents the user whose credentials have been supplied in the API request. The Authenticator class is responsible for the actual authentication of the username and password. The designed API uses role-based authentication and the Authorizer class, used here, performs the role matching and decides if a user is allowed to perform a certain action.

Dropwizard then uses theses classes to authenticate and authorize any user trying to make a request on the API. To invoke this behavior the @AUTH annotation is used in conjunction with the roles annotation, @RolesPermitted, to authorize only certain roles. This is in stark contrast to the @PermitAll annotation which allows all roles access.

This pattern and implementation across multiple client-types would be possible. Because the RESTful API is not a protocol it should be transparent to the client. For example, if the API was to be implemented on XPOX or PS4, as long as those platforms could formulate the request and receive the response no changes would be needed on the server.

The *gameauth* service would need some additional work, however. Usernames of guest, user, admin are of little use and therefore an interface to allow new usernames/accounts and their associated roles to be added. Moreover, these actions, adding users, and then assigning roles also needs to be controlled – you wouldn’t want a guest to be able to make an admin account for themselves. This might be done through an API but does pose a “juicy target” for someone with nefarious intentions. Instead, a user interface to the backend to control enrollment and entitlements behind multiple layers of security might offer a better solution.

**References**:

Unknown Author. (n.d.). *Dropwizard – BasicAuth Security Example.* HoToDoInJava. Retrieved January 29, 2022 from <https://howtodoinjava.com/dropwizard/dropwizard-basic-auth-security-example/> .

Unknown Author. (2019, November 19). *Client-Server Model*. GeeksForGeeks. Retrieved January 29, 2022 from <https://www.geeksforgeeks.org/client-server-model/> .

Unknown Author. (2021, October 19). *What is REST*. RESTfulAPI.net Retrieved January 29, 2022 from <https://restfulapi.net/> .

Unknown Author. (2020, May 8). *What is a REST API*. RedHat. Retrieved January 29, 2022 from <https://www.redhat.com/en/topics/api/what-is-a-rest-api> .