The studying materials and assignment in this module helped establish my understanding about Maven projects as well as REST style API regarding how different classes interact with each other. One thing that stands out amazing me is the multiple health checks under the run function. I believe such health checks assist the application with running smoothly and prevent any exceptions or errors from being generated. The specific Maven project model helps to align details about software by providing text codes regarding build number, release number, dependencies, parent, etc. Such model also increases efficiency because if the client has a new official public release, then by using Maven model only one line of code needs to be modified and rest will stay the same. By doing so, Maven model also ensures the stability of applications.

Before learning about Maven projects, the only place that I have used the “@” notation was “@Override”. In a Maven projects, the “@” notation has a variety of usage such as “@PermitAll” or “@RolesAllowed()”. From the server side, such notations provide direct visualization of whom is being permitted to which function. In case if errors occur, such notations can also aid with debugging and resolving any programming issues. Each “@” notation represents a specific restriction for different clients. Some will have access to a list and its details, some will only have access to the list, while others have no access at all. To perform such distinctive activity properly, the application must understand who the user is and whom to communicate to and which set of code should be processed. The server side is also responsible for creating and registering new clients. To do so, the server must know whether an account has already existed and to make sure that all new accounts being created are unique and properly registered into the server.

The client side is more complicated than the server side. In order to achieve a successful role distinction, the program must be properly denotated, coded, built, and processed. First of all, all of the “@” notations need to be placed properly. Secondly, the code block below the “@” notations need to match each role. Thirdly, the project needs to be built with dependencies and parent to ensure proper hierarchy. Lastly, when running the application, appropriate arguments should be parsed. To add more users, the GameAuthenticator class needs to be modified and more types of users could be added into the immutable map. If we were to just simply add more users based on the three primary types: guest, user, admin, then the application will utilize the GameAuthApplication class to register the user and then the user will be categorized into one of the three types. Some other features that I can include in the game app is sound effect. I am thinking to add different background music and sounds based on different scenes. If such application is to be hosted on different clients or platforms, such Maven project should still be able to carry out the main functions. One thing that we need to think about is how the new clients or platforms decipher byte code and how Maven fits into the “new environment”. Different games on different clients have different focuses, but I believe that with Maven model we can at least obtain the basic idea of game application development.