

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.30j0zll)1

[**Table of Contents**](#_heading=h.1fob9te)2

[**Document Revision History**](#_heading=h.2et92p0)2

[**Executive Summary**](#_heading=h.tyjcwt)3

[**Design Constraints**](#_heading=h.3dy6vkm)3

[**System Architecture View**](#_heading=h.1t3h5sf)3

[**Domain Model**](#_heading=h.4d34og8)3

[**Evaluation**](#_heading=h.2s8eyo1)3

[**Recommendations**](#_heading=h.3rdcrjn)5

## [Document Revision History](#_heading=h.2et92p0)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 03/17/2021 | Tyler Owens | First Revision - Executive Summary & Design Constraints |
| 1.1 | 03/29/2021 | Tyler Owens | Second Revision - Evaluations |
| 1.2 | 04/15/2021 | Tyler Owens | Third Revision - Recommendations |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_heading=h.tyjcwt)

The Gaming Room wants a web based client to serve their game, Draw It or Lose it, to multiple platforms. Draw it or Lose it is a multiplayer party game and the client will need to be able to serve an instance to multiple different clients.

## [Design Constraints](#_heading=h.3dy6vkm)

-Need to create tools that can serve the different environments game need to serve users in a way that their client can decipher or serve the game in a different way for different users

-Need to make the game function independent from the served environments game client needs to be running independently from the users so it can serve them all at the same time

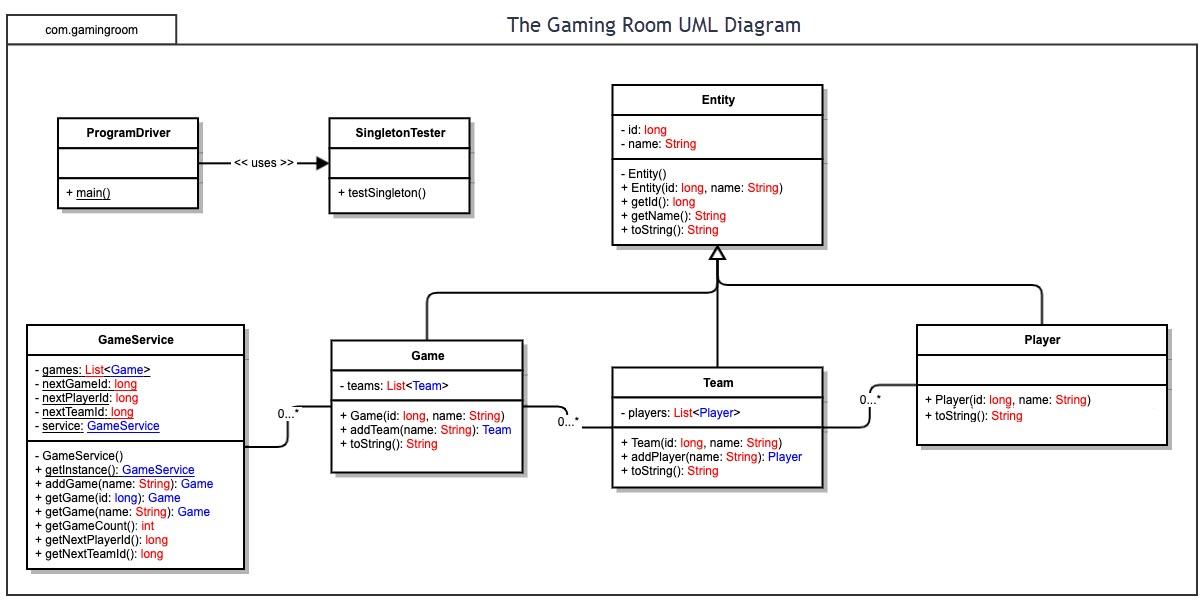
-Game needs to be running on a server, meaning server or cloud infrastructure to run games needs to be scaled to the expected number of games or users.

## [System Architecture View](#_heading=h.1t3h5sf)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_heading=h.4d34og8)

This is a mockup model for how the service could exist. The main provider is described by the GameService class that employs a Singleton design. The GameService class will contain all of the Entities internally. The Entities will all be stored in iterable lists inside their container(Games in GameService, Teams in instances of Game, Players in instances of Teams). GameService will control the id iteration of all of the different Entities to insure unique ids. There is also a Driver and a Tester class used to test these basic structures. This design should allow for seamless access from different environments.

****

## [Evaluation](#_heading=h.2s8eyo1)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | Mac PC’s would be poor for a server, they are expensive for the hardware and power they have and don’t provide good scalability | Linux servers are pretty universally used for web services so this might be our best choice (also linux is free!) | Windows servers could be a good option for hosting, would be able to meet our hardware needs, we would need to pay for licensing though | Mobile Devices do not have enough power to host our application, unless we were living in a world where all those botnets in movies are real |
| **Client Side** | If we program it in Java, a desktop app should run the same on any pc that can run the JVM so minimal changes between all PC applications/no extra development time. If we develop in other languages we will have to program for each different type of pc operating system which would improve performance but is it worth the cost? | Same note about Java, as long as we solely use it we can keep the application consistent across operating systems that can run the JVM, coding in a different language could improve performance but would cost more development time. | Same note about Java, as long as we solely use it we can keep the application consistent across operating systems that can run the JVM, coding in a different language could improve performance but would cost more development time. | Mobile devices are going to need a considerable amount of time to create a good user interface (It won’t be similar to the UI’s on desktops). And we need to consister how many different mobile operating systems we want to program for IOS, Android, etc. |
| **Development Tools** | As stated above we are going to use Java to program all pc client side applications simultaneously. Developers should be encouraged to use eclipse but can use their IDE of choice. To work together we will use a git repository | Again, it is a Java program so all pc apps should function the same. Using eclipse is free (yay) but we may need to pay for repository space. We do need to be able to set up servers on the linux systems so that we may test them so we will need addition linux machines for server testing | Also should be programmed in java with eclipse as a good free option for the IDE. Hopefully no additional work will need to be done for the various desktop applications | Hopefully we can use the java backend client side code but any UI code will need to be written for the mobile environment. Also we will not be able to program in java for IOS so we will need programmer(s) fluent in Objective-C or Switch to be able to serve the game to apple phone users. Testing devices will also be needed to test the code. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: I recommend we use AWS as the operating platform for our web servers. Cloud servers can be tricky to get up and running, but once they are implemented they are very scalable and we will only pay for the processing power that we use. We will need programmers familiar with AWS but because it is such a popular platform there are many development resources for it and plenty of trained programmers familiar with building on the platform.
2. **Operating Systems Architectures**: AWS will require some overhead to get started (not as much as traditional in-house servers) to get the code ready to be hosted on the cloud. But, once we have completed the initial development overhead, we will not need to have any physical servers. This will also allow us to pay only when we need to scale and not have to worry about scaling architecture.
3. **Storage Management**: Cloud storage again provides a cheap way to store the files we need online. We should store in a RAID configuration that allows for as many reads as possible. We will only be writing to the storage when we are changing the program's capacity. Security could be a concern but everything we will be storing on the cloud will need to be accessed and downloaded by the clients.
4. **Memory Management**: To manage the memory for the server we will be using REST as our communication technique between the games and the server. This will allow for stateless communication between the client and the server, so that the server has to store as little as possible in its memory (only the things that need to be communicated between clients). As far as client memory management, the client side code shouldn’t be too heavy, mainly consisting of menus and platform specific UI. The clients will download the needed resources for each game before the start of the game (they will need to be connected to the servers to play, so downloading compressed images shouldn’t take too long). This would only become a problem with severely poor internet access.
5. **Distributed Systems and Networks**: Since this is a multiplayer game, the players will need to have a stable internet connection in order to be able to play. Using AWS, our service can be distributed from multiple servers and should lead to minimal server side outages. We will be unable to control if our players have a stable internet connection and games will continue if one or more players are unable to connect. Because we are going with a REST-style communication, players can reconnect at any time and the stateless service should be able to handle it. Again because we decided on using REST services as long as the clients communicate the same way to the servers, it shouldn’t matter what types of devices are connecting for any game as the service should be able to handle any combination of client types.
6. **Security**: Security is always a concern, especially when using a cloud environment that can potentially be accessed from anywhere. Best programming practices should be used, such as encapsulation, to keep data secure. Fortunately the only sensitive data we would be storing is username and passwords. Any username and password handling should be encapsulated and passwords should be stored as hashes so that even if the data does get stolen, it would be useless. None of the other data would be sensitive, so we need to protect from someone writing over our data but we do not have to be too concerned with the data being stolen.