

Win, Lose or Draw

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

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

[**Table of Contents**](#_heading=h.30j0zll) **2**

[**Document Revision History**](#_heading=h.3znysh7) **2**

[**Executive Summary**](#_heading=h.2et92p0) **3**

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

[**System Architecture View**](#_heading=h.3dy6vkm) **3**

[**Domain Model**](#_heading=h.1t3h5sf) **3**

[**Evaluation**](#_heading=h.2s8eyo1) **4**

[**Recommendations**](#_heading=h.3rdcrjn) **6**

## [Document Revision History](#_heading=h.3znysh7)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.2 | 06/14/2020 | Camilo Hoyos | Updated recommendations section |
| 1.1 | 06/01/2020 | Camilo Hoyos | Updated Evaluation section |
| 1.0 | 05/19/2020 | Camilo Hoyos | Initial doc with Executive Summary, Design Constraints, and Domain Model |

**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.2et92p0)

The Gaming Room company does not have staff that is knowledgeable in setting up a game environment. They want to deploy a web-based version of the game. There are some rules around how the game is played that need to be enforced.

## [Design Constraints](#_heading=h.tyjcwt)

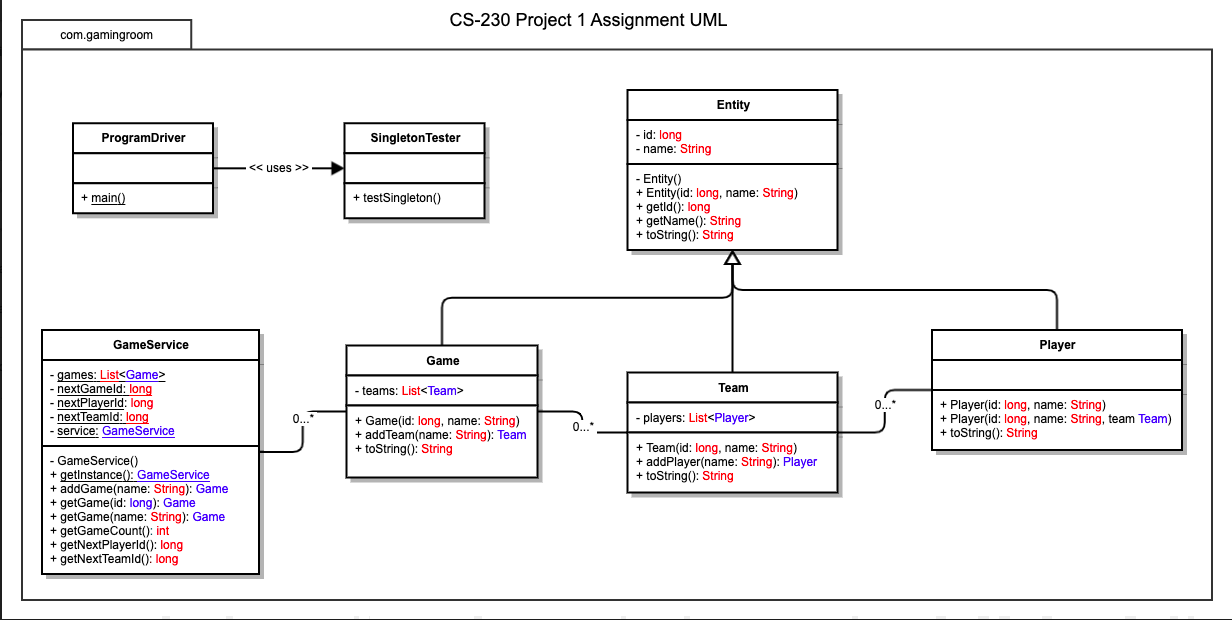
* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [System Architecture View](#_heading=h.3dy6vkm)

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.1t3h5sf)

The Entity class now serves as a base class to the Game, Team, and Player classes. This is because they all share ID and Name. GameService has a zero to many association with Game. Game has a zero to many association with Team. Player has a zero to many association with Player. The Game having a zero to many association with Team fulfills the requirement ‘Game will have the ability to have one or more teams involved’. The team having a zero to many association with Player fulfills the requirement of ‘Each team will have multiple players assigned to it’. The singleton creation pattern that has been designed for GameService ensures only one instance of the game can exist at any given time. The iterator design pattern ensures team names and game names are unique.

****

## [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** | PRO   * Reliable with Mac based development * Interface with Mac tools   CON   * Vulnerable to setuID exploit | PRO   * Cost efficient * Open Source * Has multiple control panels   CON   * No access to MSSQL, VBS, C#, MS Access * Vulnerable to setuID exploit | PRO   * Easy to learn/user friendly to maintain server   CON   * No access to SSH or apache modules * Only 1 control panel | No real known benefits to mobile hosting.  CON   * Nearly no configuration/tools |
| **Client Side** | Expensive hardware is a disadvantage. However, they are known for being able to work on all three platforms using homebrew. The ease of developing against windows, mac, and linux is a time saver. | Known to take a lot of time to configure, but most languages are easily accessible through ports. | Only seems to particularly beat out other competitors if development is C# or .NET language/framework. | User experience is going to be the heaviest hit. Since this is on a different level than desktop, lots of time would be invested to convert. If Swift/Java are not used then workarounds will need to be executed to have client side mobile apps. |
| **Development Tools** | Currently Eclipse and IntelliJ are known IDEs for Java. Notable deployment tools for Java: Jenkins/Team City, Bamboo. | Currently Eclipse and IntelliJ are known IDEs for Java that are shared with other OS. Notable deployment tools for Java: Jenkins/Team City, Bamboo. | Currently Eclipse and IntelliJ are known IDEs for Java that are shared with other OS. Notable deployment tools for Java: Jenkins/Team City, Bamboo. | Would have to understand the deployment behavior for Apple store and Google play. Xcode is the known IDE at the center of apple development. Currently Eclipse and IntelliJ are known IDEs for Java that are shared with other OS. Bamboo is a deployment tool that integrates directly with Apple Store and Google Play. |

## 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**: For development/server, it does seem as though Mac is favored for the ease of developing on multiple platforms. There is a high cost associated with this through hardware though. The way I see it, if you have an experienced team who is comfortable developing on windows/linux you can find cost effective ways to develop and still meet multi-platform needs. If you have an inexperienced team, then spending the additional investment on Mac will ease the burden of developing on multiple platforms. My personal recommendation would be to pursue a linux based server. Linux appears to be more ‘future’ proof, if requirements change or business pivots, the flexibility of linux allows for that change to be managed.
2. **Operating Systems Architectures**: Regarding systems architecture since multiple users will be in the same game, a three-tiered approach seems feasible. Since 4 players (clients) will be in the same game they will share the same section of memory. They will share the same image as well. so the database layer stores the images of which to render, the application server will manage the ‘games’ or the images more specifically. This middle tier of the architecture is where all of the efficiency gains are since players will share the same application server/memory.
3. **Storage Management**: Since we know the need to manage 1000 active games at a time we need to be able to manage that amount of throughput for storage. At peak it is possible to request upwards of 1000 images every 30 seconds (game rounds). This would be pulled from storage and then loaded into memory. This means we need to be able to handle roughly 2.6GB/sec. There are SSDs that can have upwards of 10GB/sec throughput, so there is certainly an SSD available that can host the need. Not all images are exactly 8MB so having additional throughput will ensure a clean experience. Additionally, since there won’t be manipulation of storage from users, it should be stored in contiguous allocation to save space. This would allow the purchasing of smaller storage, with fast throughput and save on cost. Right now memory appears to only need around 1.6GB of storage for the images alone. Not yet knowing what other files or data needs to be stored, it is hard to define what the storage size should be, but keeping this low and well organized will allow for better storage efficiency.
4. **Memory Management**: Similar to storage management we have some numbers to work with. We know that at most there will be 4000 players active in 1000 games. Since 1000 games will be hosted and it is possible to have them all active at once with 1 image, we’re looking at roughly 8000MB or 8GB of RAM to be consumed by images alone. Again, without further understanding what other actions users can take or how business plans to allow users to interact within the game this initial measurement is a starting point. We know players, games, and teams need to be recorded but this is all simple data to be recorded and should not vastly increase the amount of memory needed. Depending on future roadmap items, it is possible more RAM may be needed. Similar to storage management as well, since a lot of this data is defined up-front we can execute best fit and compaction to manage RAM accurately. This ensures we don’t have a lot of chunked blocks of data that cannot be utilized.
5. **Distributed Systems and Networks**: It would seem to best host it on a linux server due to its compatibility and flexibility. Then for connectivity ensure that actions occur on the server side so that if connectivity is lost, final actions can still be executed from the server system. Losing connectivity from the other direction would be more detrimental as the ‘game’ would not effectively communicate to the other players.
6. **Security**: Ensure that databases are restricted and do not allow for admin level access. Validate all credentials in any CRUD operations. We also talked about file based validations and how the server can enable read-only based operations to ensure no manipulation can occur. Since images aren’t intended to be changed, forcing read-only would create a safer application. In addition, we experienced CRUD operations in which all GET/POST/PUT/DELETE from HTTP were validated against the user role. This same process should be executed to ensure no users manipulate data where they should not be allowed to. As an example, admins could manipulate users, but other players/users would not be able to manipulate other players/users.