

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/18/2021 | Joshua Morris | Preliminary architecture completed for class structures of the GameService 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](#_sbfa50wo7nsh)

The Gaming Room is designed to meet the need of the Have it or Lose it game for an organizational structure. The company requires a multiplatform, web-based application that can create, access, and manage player and team accounts across multiple game instances, while maintaining unique identifiers for each. The proposed solution must allow administrators to access this information, while simultaneously guarding against accidental corruption of data, or duplication of user accounts. The Java language was decided upon as the best fit for this situation, due to its robust class structure and strong web-based capabilities.

## [Design Constraints](#_2et92p0)

1. Since this will be a server-based web app, it must be capable of creating and tracking multiple instances of its base entities, while maintaining connection to a single unifying hub.
2. Created instances must have unique identifiers, which should be tracked and enforced by the central server.
3. Creation of empty instances should be prohibited, in order to avoid unnecessary traffic to the server.
4. The central server must manage multi-level entities, that is, multiple games, each of which contains teams, which in turn contain players.
5. The server should not allow creation of a player or team instance if one of that name already exists.
6. The server hub must be unique; creation of a second server instance is prohibited.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

As can be seen from the diagram below, the classes representing Players, Teams, and Games are all built upon the base Entity class. This structure of inheritance allows for more efficient management of the entities by the driver classes, as well as ensuring that all entities within the application adhere to the same basic design structure. It will also prove useful as development proceeds, as any attributes or functions common to all three classes can simply be added to the base class, rather than implementing them separately.

A second feature to note is the presence of ‘private’ default constructors in the Entity-based classes. This feature ensures that entities are not created with null values (in order to create an instance of the class, the user must supply a set of initial values). Not only does this prevent coding accidents (since attempting to create an empty instance will result in an error), it also helps prevent server failure that could occur if a massive number of empty instances were created via an exploit.

Finally, note the unique structure of the GameService singleton class. The structure is designed to essentially instantiate itself, after which any calls to *GameService* will return that instance. So long as the program is running, this is the only instance of the class that can exist in memory. This setup is required to guard the status of the nextGameId, nextPlayerId, and nextTeamId counters. By ensuring that only one instance of the server class can exist, we ensure that the counters will be accessed and incremented correctly for each entity.

****

## [Evaluation](#_2o15spng8stw)

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 OS is generally consistent between desktop and mobile applications, which make it easy to port apps between them. Additionally, Macs are overall more secure and consistent than other systems; a fact which can help smooth development of web-based clients. The downside is that iOS uses a lot of proprietary technology, and is generally incompatible with many open-source and third party resources. | Linux is a lightweight OS with excellent stability and fairly simple design requirements. Linux has strong security features and supports most Java functionality. The downside in a browser-based server application is that Linux browsers are often open-source third party applications, and may exhibit unexpected behavior client-side | Developing for Windows is made easier by the fact that it is currently the most widely used OS There are a huge number of tools and resources available to support the development process. The drawback is that Windows often struggles with stability and security features, especially with regard to network security. | Developing server-based apps for mobile OS is fairly simple; most mobile SDKs integrate JavaScript and server functions, as web-based functionality is fairly standard for the mobile platform.  The primary consideration in this case will be data encryption and ensuring that the application is able to function over the comparatively limited bandwidth offered by mobile service (as opposed to landline internet typically used for desktops). |
| **Client Side** | Similarity between desktop and phone applications allows the two clients to be developed side by side. iOS is generally consistent in structure and hardware integration, so fewer special considerations are needed in terms of back-compatibility or version updates. | Linux is open-source, which assists the development process. The downside is that there are fewer tools/resources available for Linux, since it is less popular. This also tends to diminish the return on investment of development for Linux systems. | Windows has no mobile equivalent, so a javascript-based web app will serve best here. Windows versions are not always consistent, so care must be taken to keep the app updated, and backwards compatibility may not always be possible. | Developing for mobile OS is facilitated by a large number of integrated SDKs that make it relatively easy to build cross-platform apps from the ground up.  However, if the current Android app uses open source components, some of them may have to be changed, as iOS generally does not support as many open source parts. iOS also lacks many of the customization features that Android supports. The same applies to UI and controls. |
| **Development Tools** | **Languages:**  Java, Javascript  **Tools:**  Eclipse IDE/  Netbeans IDE  Appcelerator  Maven  Dropwizard | < **Languages:**  Java, Javascript  **Tools:**  Eclipse IDE/  Netbeans IDE  Appcelerator  Maven  Dropwizard | **Languages:**  Java, Javascript  **Tools:**  Eclipse IDE/  Netbeans IDE\  Appcelerator  Maven  Dropwizard | **Languages:**  Java, Javascript  **Tools:**  Buildfire.js  JQuery Mobile  Appcelerator |

## 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**: Based on the game’s requirements, the need for stability, and industry comparisons, we recommend Amazon EBS (Elastic Block Storage) as the best option. While the service is more expensive per GB, it offers fast speeds and dedicated Solid State Drives to facilitate rapid data access. We are making this recommendation based on the fact that the game will require the ability to store, access, and swap out thousands of photos at an extremely rapid pace, and EBS’s SSD-backed storage is specifically designed to reduce read/write time and facilitate seamless access. For Draw it or Lose it, the recommended division is EBS Provisioned IOPS SSD (io1).
2. **Operating Systems Architectures**: The service uses small volume SSDs (1 – 16 TB) to facilitate the quickest possible access (Draw it or Lose it is unlikely to require anything larger, so the size is not an issue). While it lacks the sub-millisecond latency of the higher tiers, the single digit millisecond rate should more than meet the requirements of the client’s game. The structure uses block storage to facilitate multi-source access. While this is less efficient than object storage at handling rapid changes in data, the game’s image files should not suffer, since they will remain static for some time before being changed.
3. **Storage Management**: Amazon offers a service called data lifecycle that automatically manages data, applying updates and backing up snapshots according to a customizable plan. Storage can be automatically allocated (and billed) on a day-to day basis. This means that, should the game require updates or expansion, there would be no interruption of service while the memory blocks are changed.
4. **Memory Management**: EBS is specifically designed for applications with high transactional workloads. The servers are capable of processing 160,000 IOPS (Input/output operations per second) per instance; while it is unlikely that the game would require more than this, a second server instance could be created to manage additional traffic flow.
5. **Distributed Systems and Networks**: Amazon’s EBS servers are designed for cross-platform compatibility, and provide dedicated backup servers for components in order to prevent outages. These duplicate servers are provided at no additional charge. The service guarantees 99.8% reliability and performance according to provided specifications, and is backed by Amazon’s existing history of credibility. The guarantee, combined with the ability to maintain data snapshots as backup, allows them to offer a 99.8% guarantee of reliability.
6. **Security**: EBS is designed using Amazon’s own security protocols, including data encryption, key tracking, and usage history. The servers allow users to set encryption levels for specific data; in the case of the game, image files would not need to be as secure as user data or payment information. There is no additional charge for encryption on any level.