

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0  
  
Gordon Price

11/12/2023

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/12/2023 | Gordon Price | No revisions were done, we established and brainstormed a necessary structure to tackle the client’s project and goals. |

**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)

Designing software for a gaming application within a web-based distributed framework poses a significant challenge. In response, my proposed solution involves the creation of a Java application rooted in the principles of object-oriented programming. I will develop key classes, including GameService, Game, Team, and Player, to facilitate smooth interactions in the management of games, teams, and players. Employing a modular design, the foundational Entity class will consolidate crucial attributes like ID and name across all entities. This design approach allows me to meet software requirements efficiently, guaranteeing the development of a resilient and operational gaming application.

## [Design Constraints](#_2et92p0) The design constraints influence how we build the application. They make us think carefully about things like how the network works, how to keep things secure, using Java, designing things with objects in mind, making sure the application can handle a lot of users and runs smoothly, and how the user interface looks and works.

Web-Based Distributed Environment:

The game application is required to function in a web-based distributed environment, demanding accessibility over the internet and the ability to support multiple concurrent users. This constraint necessitates meticulous design to address potential network latency issues and integrate robust security measures.

Java Programming Language:

The software development is mandated to be exclusive to the Java programming language. This constraint confines the technology stack to Java-based frameworks, libraries, and tools, imposing strict adherence to Java coding conventions and best practices throughout the development lifecycle.

Object-Oriented Design:

The application's architecture must strictly adhere to object-oriented design principles, emphasizing modularity, reusability, and maintainability. This constraint requires the creation of classes with proper encapsulation, inheritance, and polymorphism. Moreover, the application should integrate design patterns and abstraction techniques to foster a flexible and extensible codebase.

Scalability and Performance:

The design of the game application must be scalable to accommodate a growing number of games, teams, and players, ensuring scalability without significant performance degradation. Adhering to this constraint involves optimizing database access, minimizing resource consumption, and implementing caching mechanisms to enhance the overall efficiency of the system.

User Interface:

The application's user interface is expected to be intuitive, user-friendly, and responsive, delivering an engaging experience for players. To meet this constraint, the design should incorporate web technologies such as HTML, CSS, and JavaScript, creating a visually appealing and responsive user interface that facilitates efficient interaction with games, teams, and players.

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram provides a visual overview of how various components collaborate within our game application. Classes such as GameService, Game, Team, and Player are intricately connected to fulfill the software's requirements.

At the heart of this connection is the Entity class, acting as a shared foundation for all other classes by offering common attributes like ID and name. This cohesion is achieved using the inheritance concept, enabling other classes to inherit these shared attributes. Utilizing the Entity class ensures a tidy codebase without redundancy.

The GameService class assumes the role of the primary manager, overseeing all games. It keeps tabs on the games and provides information, such as the total number of games. To maintain a single managerial instance, we implement the Singleton pattern. Additionally, a dedicated class, SingletonTester, assesses the proper functioning of this managerial approach. Meanwhile, the Game, Team, and Player classes each play a distinct role in representing specific games, groups within a game, and individual players, respectively. This structured approach, known as composition, involves building larger entities (e.g., the Game class) by assembling smaller components (e.g., teams and players), akin to constructing a tower with diverse building blocks. The hierarchical arrangement allows for effective management of games, teams, and players in the application.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## 

## [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 serves as a web server, less popular than Linux or Windows. It offers a secure Unix environment, with advantages such as stability and support for web applications. Weaknesses include limited hardware options and potential higher costs compared to Linux alternatives. | Linux is a favored choice for hosting web applications, boasting various distributions tailored for servers, ensuring stability and scalability. Its advantages include cost-effectiveness, high customizability, and a broad selection of server software. However, setting up and managing Linux may demand more technical expertise than Mac or Windows alternatives. | Windows Server is frequently employed for hosting web applications, particularly those built with Microsoft technologies. It excels in integrating with Microsoft tools and technologies, providing various web server options such as IIS, and offering robust support for .NET applications. However, drawbacks include potentially high licensing costs and may not be as well-suited for open-source software development. | Mobile devices usually don't host web applications; instead, servers handle mobile app data, and Mac, Linux, or Windows can serve as backends. Advantages include reduced latency, cost-efficiency for specific cases, and straightforward deployment for small-scale applications. However, weaknesses encompass limited resources causing performance issues, scalability challenges, reliability concerns such as battery drain and hardware failures, security risks, network dependencies, potential bandwidth limitations, and complexities in maintenance and management. |
| **Client Side** | Developing for Mac clients usually entails using Apple's Xcode development tools, with moderate costs and a need for expertise in Swift and Objective-C. | Considerations for Linux software development depend on the distribution and desktop environment. Costs are generally low, but expertise may be necessary for distribution-specific nuances. | Considerations for developing software on Windows clients often entail using Visual Studio. Costs can vary, and expertise in .NET languages such as C# may be necessary. | Considerations for mobile device development necessitate platform-specific approaches, with costs contingent on the number of targeted platforms. Expertise in languages such as Swift for iOS, Kotlin for Android, or cross-platform tools like Flutter may be necessary. |
| **Development Tools** | Xcode is the main IDE for Mac applications, supporting programming languages such as Swift, Objective-C, and C++. | Linux accommodates a diverse array of programming languages, and prominent integrated development environments (IDEs) include Visual Studio Code, IntelliJ IDEA, and Eclipse. | Primary IDE for Windows development, Visual Studio supports languages including C#, C++, and others. | Programming languages and tools for mobile development vary by platform. For iOS, Xcode with Swift/Objective-C is standard, while Android commonly employs Android Studio with Kotlin/Java. Cross-platform tools like Flutter, utilizing Dart, offer a streamlined approach for development across both platforms. |

## 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**:   
     
   Firstly, when planning the game, it's crucial to select an operating platform that facilitates cross-platform development, considering the game's multi-platform nature. For instance, web-based games can leverage HTML5, CSS, and JavaScript, ensuring compatibility across various platforms like Windows, MacOS, Linux, Android, and iOS. Moreover, opting for a platform with an extensive development community offers advantages such as collaboration, knowledge exchange, and access to a wealth of resources. That being said, I would recommend using Microsoft Windows Server 2016 because it’s a powerful and versatile OS that can be used to host a large variety of applications.
2. **Operating Systems Architectures**:   
     
   The Windows Server 2016 operating system features a multi-layered architecture, with key components including the kernel, which serves as the core responsible for managing hardware and providing essential services to other layers. In the user mode layer, applications run independently, isolated from the kernel mode layer to safeguard against potential malicious code. Additionally, drivers, acting as intermediary software, establish a layer of abstraction between the operating system and hardware. Their role is crucial in facilitating communication between the operating system and hardware without necessitating a detailed understanding of the hardware's intricacies.
3. **Storage Management**:   
     
   Because the game is online, it's important to store its data in the cloud for easy access across different platforms. Picking a reliable, secure, and scalable storage system is essential for the growing number of users. Good memory management is also crucial for smooth gameplay without errors. Choosing an architecture that supports efficient memory management helps prevent issues like memory leaks, enhancing the overall game quality.
4. **Memory Management**:   
     
   For games accessible on different platforms, a distributed system is needed to manage traffic and ensure smooth connections between users. Choosing an architecture that supports load balancing, auto-scaling, and fault tolerance is crucial for handling many users and ensuring a smooth game experience. Windows Server 2016 employs a mix of techniques like paging, memory mapping, and virtual memory to stay on top of how memory is used and to guarantee that applications get the memory they require.
5. **Distributed Systems and Networks:**

Windows Server 2016 is tailor-made to bolster distributed systems and networks. With a range of features, it simplifies the development and deployment of distributed applications. Among these features are Remote Procedure Calls (RPCs), enabling seamless communication between applications across a network. Distributed File Systems (DFS) facilitate access to files stored on remote servers. Additionally, Web services open avenues for applications to communicate through the web, creating a well-connected and collaborative environment.

1. **Security:**

Windows Server 2016 doesn't mess around when it comes to security – it comes armed with a full suite of protective features for your data and applications. These include user accounts and passwords, serving as the gatekeepers to authenticate users and manage resource access. Data encryption acts as a shield, ensuring sensitive information stays safe from unauthorized access. Firewalls stand guard, blocking any unauthorized access attempts to the network. And if that's not enough, Intrusion Detection Systems are on the lookout, ready to detect and alert administrators to potential security threats.