

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

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

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

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

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations**](#_26in1rg) **11**

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/16/23 | William Fishburn | Changes made to Executive Summary, Requirements, Evaluation, Design Constraints, and Recommendations. |

## [Executive Summary](#_35nkun2)

The software design challenge at hand involves developing a web-based gaming application for The Gaming Room, who currently have an Android app; they desire to expand their presence by offering their gaming experience on the web. The key objective is to create a robust and user-friendly web app that fulfills their specific software requirements while delivering an immersive gaming experience across diverse devices and platforms.

To address this challenge, our proposed solution involves meticulous planning and documentation regarding the architecture, features, and functionality of the web-based app. Using our expertise in web development, we will design a gaming application that allows participation of one or more teams, assigns multiple players to each team, and guarantees uniqueness of game and team names.

## Requirements

* *A game will have one or more teams.*
* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique so users can verify if a particular name is in use when a team name is selected.*
* *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.*

## [Design Constraints](#_1ksv4uv)

Network Limitations: Latency and bandwidth limitations can restrict the amount of data that can be transmitted between the client and server. This constraint can affect the quality of graphics, audio, and video in the game, as well as the speed of data updates. Development efforts should focus on optimizing data transfer, compressing assets, and prioritizing critical data to ensure a smooth and responsive gaming experience within the limitations of available bandwidth.

Device and Browser Compatibility: The web-based game app needs to function seamlessly across different devices and web browsers. Variations in hardware capabilities, screen sizes, and browser capabilities can pose design challenges.

**Implications on Application Development:**

The design constraints impact various aspects of application development. These require careful consideration of strategies to overcome the respective challenges. Development efforts should focus on optimizing network communication and minimizing the impact of latency on real-time interactions. Efficient memory allocation and load-balancing techniques will be critical in enabling the application to function at scale.

## [System Architecture View](#_44sinio)

## [Domain Model](#_2jxsxqh)

The Entity superclass utilizes a key OOP concept of inheritance. It efficiently establishes a relationship between the Game, Team, and Player classes so that all three classes obtain code directly from the Entity class. Then by aggregation, the Team class has a list of Players, the Game class has a list of Teams, and GameService has a list of Games. The ProgramDriver class contains the main() method that will run the SingletonTester program.

**"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](#_z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac operates on a Unix-based operating system (macOS), which offers a stable and secure environment for hosting web applications.   It inherits the robustness and security features of Unix, making it well-suited for web hosting.  Mac hardware options are relatively limited compared to other platforms, which can result in higher upfront costs for purchasing Mac-based servers. | Linux is an open-source operating system that offers flexibility and customization options. Its source code can be modified, allowing users to tailor the system to their specific needs. Linux offers a wide range of hosting options, including shared hosting, virtual private servers (VPS), dedicated servers, and cloud-based solutions. This versatility allows for scalability and flexibility to meet various hosting needs.  Linux can have a steeper learning curve for users unfamiliar with the system. Its command-line interface and configuration files require a level of technical expertise, potentially making it more challenging for novice users. | Windows is a widely adopted operating system, known for its user-friendly interface and compatibility with a vast range of hardware and software applications. Its popularity can facilitate ease of use and familiarity for developers and users.  Windows provides a wide array of development tools, frameworks, and integrated development environments (IDEs) tailored for web application development. These tools, such as Visual Studio, enable developers to efficiently create and debug web applications.  Windows operating system licenses can have additional costs compared to open-source alternatives like Linux. This can impact the overall budget for hosting web applications, particularly for organizations with a large number of servers or high scalability requirements. Windows may offer less flexibility and customization options compared to open-source platforms. | Mobile devices are highly portable and allow users to access web applications from anywhere with an internet connection. They provide flexibility and convenience, enabling users to engage with web-based software on the go, while integrating with native device capabilities, such as push notifications, camera access, GPS, and accelerometer. These can enhance the functionality and user experience of the web application.  Mobile devices have established app stores (e.g., Apple App Store, Google Play Store) that serve as centralized distribution platforms. Hosting a web-based software application as a progressive web app or a hybrid app can allow for easy distribution and reach through these app stores.  Mobile devices typically have limited resources compared to desktop or server environments. Factors such as processing power, memory, and storage capacity can impose constraints on the complexity and performance of the hosted web application. |
| **Client Side** | Cost: Developing software to support multiple types of clients on Mac may involve costs related to Mac hardware and software licenses. Ensuring compatibility and testing across different Mac devices can also incur additional expenses.  Time: Developing software for multiple types of clients on Mac may require additional time for platform-specific development, testing, and debugging.  Expertise: Development expertise in Mac-specific technologies, frameworks, and tools (such as Xcode and Objective-C/Swift programming languages) is necessary for effectively supporting multiple client types on OSX. Familiarity with Mac-specific features, APIs, and best practices is crucial for optimal software performance and user experience. | Cost: Supporting multiple types of clients on Linux can result in cost savings due to the open-source nature of Linux. Licensing costs are typically minimal, and Linux distributions are freely available. However, costs may still be incurred for hardware, maintenance, and potential customization needs.  Time: Developing software for multiple types of clients on Linux may require time to ensure compatibility across different Linux distributions, kernel versions, and libraries. Testing and addressing compatibility issues can extend the development timeline.  Expertise: Expertise in Linux development and system administration, proficiency in Linux-specific programming languages ( C, C++, Python), and familiarity with distribution-specific capabilities are essential. | Cost: Supporting multiple types of clients on Windows may involve purchasing Windows licenses, development tools (e.g. Visual Studio), and Windows hardware.  Time: Although Windows has a mature and widely-used development ecosystem and well-documented APIs, additional time may be required for testing and ensuring compatibility across different versions of Windows, including desktop and mobile platforms.  Expertise: Expertise in Windows development frameworks (e.g. .NET) and programming languages (e.g. C#) is necessary for supporting multiple clients on Windows. | Cost: Supporting multiple types of clients on mobile devices can involve costs related to development tools, licenses (such as Apple Developer Program), and hardware for testing purposes.  Time: Developing software for multiple types of clients on mobile devices may require additional time for platform-specific development(iOS, Android), testing, and adapting the user interface to fit different screen sizes and resolutions.  Expertise: Proficiency in mobile app development frameworks and programming languages (e.g. Swift, Kotlin) is necessary for supporting multiple types of clients on mobile devices. |
| **Development Tools** | Programming Languages:  Objective-C: A primary programming language used for macOS and iOS development. It is the traditional language for developing Mac applications.  Swift: Apple's modern programming language designed for macOS, iOS, watchOS, and tvOS development. Swift has gained popularity for its safety, performance, and ease of use.  Integrated Development Environments (IDEs):  Xcode: Apple's official integrated development environment (IDE) for macOS and iOS app development. It includes a suite of tools, editors, and debuggers to support the development process.  Interface Builder: A visual editor integrated within Xcode, allowing developers to design user interfaces for Mac and iOS applications through drag-and-drop and declarative interface design. | C/C++: Widely used languages for Linux development, offering low-level access to system resources and efficient performance. However, this compiled language does not come with the same level of portability as Java or Python.  Python: A popular scripting language with a large user base and extensive library support. It is known for its readability, portability and ease of use.  Java: Highly portable, platform-independent language commonly used for enterprise applications. Linux provides strong support for Java development. | Eclipse: A versatile IDE that supports multiple programming languages and offers features like code editing, debugging, and project management.  Visual Studio Code: A lightweight, open-source code editor with excellent support for various programming languages, including Linux development.  GNU Compiler Collection (GCC): A collection of compilers, including C, C++, and other programming languages, providing a comprehensive set of tools for compiling and building software on Linux. | Programming Languages:  Swift: Apple's programming language for developing iOS, macOS, watchOS, and tvOS applications  Objective-C: A primary programming language used for iOS and macOS development, especially for mixed-language projects on Apple platforms.  Java/Kotlin: Java is a widely popular language for Android development, and Kotlin is a modern programming language that can operate seamlessly with Java. |

## Recommendations

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

**Operating Platform**: I recommend considering a web-based approach. Developing a web application allows you to target various computing environments without being tied to a specific operating platform. Users can access your app through web browsers on different devices, including desktop computers, laptops, tablets, and smartphones, regardless of their operating system.

**Operating Systems Architectures**:  
  
**Client:**

Web Browser: The client component, typically a web browser installed on the user's device, is responsible for rendering and displaying the user interface (UI) of the web application. It handles user interactions, such as clicking buttons or submitting forms.

**Server:**

Web Server: The server component hosts the web application and delivers its content to client devices. It processes user requests, executes business logic, and retrieves/ stores data as necessary.

**Communication:**

Communication between the client and server takes place through the Hypertext Transfer Protocol (HTTP), which operates as a request-response protocol. The client initiates the communication by sending HTTP requests to the server, which then responds by providing the requested data or performing the necessary actions.

Application Programming Interfaces (APIs) play a crucial role in software development. They define and expose specific functionality or data for external use, integrating with third-party services, databases, or access to platform-specific features.

**Storage Management**: Relational Database Management System (RDBMS). RDBMS systems provide a structured and organized approach to storing and managing data. Here is a popular RDBMS option to consider:

PostgreSQL: An open-source RDBMS known for its expansiveness, and ability to integrate with development frameworks such as REST API. PostgreSQL offers advanced features like JSON support, spatial data types, and full-text search, making it suitable for complex web applications.

**Memory Management**:   
  
Garbage collection/dynamic memory allocation:

The web browser manages memory to handle the execution of web applications. The browser's JavaScript engine performs automatic garbage collection, reclaiming memory that is no longer in use by the application. This process identifies and frees up memory occupied by objects that are no longer referenced by the application.

**Distributed Systems and Networks**:   
  
Network Connectivity:

The distributed software system relies on a network infrastructure to establish connectivity between devices and platforms. This can include local area networks (LANs), wide area networks (WANs), or the internet.

Protocols and APIs:

Communication between platforms is facilitated by using protocols and APIs (Application Programming Interfaces). Common protocols include HTTP/HTTPS for web-based communication, TCP/IP for reliable data transfer, and MQTT or AMQP for messaging-based communication. APIs define the interfaces that enable different components of the distributed system to interact and exchange their data.

Distributed System Components:

The distributed software system consists of various components spread across different platforms. These components may include servers, clients, databases, and other services. Each component performs specific functions and collaborates to achieve desired communication and functionality.

Dependency Management:

Dependencies between components within the distributed system are managed to ensure smooth operation. This includes managing software dependencies(package management, database integration), version control, and ensuring required libraries are accessible.

**Security**:   
  
Secure Data Transmission:

Utilize secure communication protocols such as HTTPS (HTTP over SSL/TLS) to encrypt data transmission between the user's device and the server. This ensures that sensitive information remains encrypted during transit, protecting it from unauthorized access.

Authentication and Authorization:

Implement robust user authentication mechanisms to verify the identity of users accessing the platform. This may include techniques like username/password authentication, two-factor authentication (2FA), or integration with external authentication providers (e.g., OAuth). Additionally, enforce proper authorization controls to ensure users can only access the data and functionalities they are authorized to use.

Data Encryption:

Employ strong encryption techniques(SHA-512, RSA, TLS) to protect user information when stored on servers or transmitted across different platforms. Encryption ensures that even if data is compromised, it remains unreadable without the encryption keys.