

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 01/28/2024 | John Schatzl | <Brief description of changes in this revision> |

**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 seeking to expand its Android app, “Draw It or Lose It” into a multi-platform web-based game. Our team is tasked with creating a software design that sees to the simplification of this transition and ensures that a seamless user experience is maintained across every platform. Suggesting a solution that uses many modern web technologies to create a responsive, engaging and scalable online gaming platform. Maintaining game integrity through unique game instances, names and supporting a clean and functional interface.

## Requirements

This software design will address some key requirements:

* Multi-platform compatibility to reach a wider player base.
* Web-based deployment for simple access and maintenance.
* Unique identifiers for each player and game.
* Capability to support multiple player and teams in a single instance.
* Emphasis on maintaining existing game mechanics and user interface.

## [Design Constraints](#_2et92p0)

The implementation of a web-based game application faces many constraints to the team. First, is that of browser compatibility. Many modern browsers support the game, but there are many others that may have issues with some technologies. Second, is that of network and internet dependency. As a web based game, you inherently have to depend on network latency and internet connectivity. The lack of offline features may pose as a hindrance to the goal of the project. Finally, data synchronization between clients and users could pose a significant constraint to the design. Ensuring that the user and game data is consistent across the same user on different clients or vice versa, is paramount to the longevity and interoperability of the game. These constraints can add many labor hours and resources to the project as implementing these require resources that may not have had any necessity in a solely Android based environment. But the centralization of this game as a web-based platform brings with it cross-platform compatibility and all maintenance in one place.

## [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)

The ULM diagram for The Gaming Room application is representative of a web-based deployment of the game Draw It or Lose It and includes many classes and properties. Starting with ‘Entity”, a base class with common attributes ‘id’ and ‘name’ suggests inheritance among all children and entities in the domain. ‘GameService’ is a singleton class due to its private constructor and static method ‘getInstance’ which ensures only one instance manages the logic and data of the game. ‘Game’ representing the game instance, has a unique ID, name and a list of ‘Team’ instances. Speaking of which, ‘Team’ corresponds to a group of players within the game, housing a unique ID, name and list of ‘Player’ instances. Finally, the ‘Player’ instance defines the individual players participating in a team with a unique ID and name.

There are many object-oriented principles employed within the diagram. Firstly, inheritance plays a large role in the diagram. Instances like ‘Game’, ‘Team’, and ‘Player’ all inherit from the ‘Entity’ class as they share common attributes.

Another principle shown Is the idea of encapsulation. Some attributes that are kept within private within classes, suggest the use of getters and setters for accessing and manipulating them.

Used in the ‘GameService’ instance, a singleton pattern is used to ensure that only one instance controls logic and data, which meets the requirement of a single game instance in the memory.

Finally, aggregation, is used in almost every instance, describing a ‘has-a’ relationship between instances and their properties.

**"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** | Known for its GUI and stability, has an intuitive interface, stable and secure. Less commonly used, software and hardware can be very expensive. | Open-source, widely used among many services for servers and offers a wide range of server software. But, requires expertise and can be very verbose. | Widely used, easy integration with other Windows Machines, but has a higher vulnerability to malware and incurs licensing costs. | Offer portability and ease of access, but need to have lightweight loads. Hardware provides less power but can be more efficient. Robust security features. Use-case for server side development is very limited. |
| **Client Side** | Cost for Mac clients can be expensive due to their proprietary software and tools. Development time would take longer due to strict design guidelines and approval processes through the App store. | Cheapest option, but requires more work due to it’s various distributions. Requires knowledge of Linux-specific programming and packaging for distributions. Managing the different distributions for compatibility can be time-consuming. | Licensing costs for software, but it can streamline the development through various technologies. Knowledge of C# and .NET required. Managing different versions of Windows can be difficult, but manageable. | Depending on the platform, costs can vary. iOS guidelines and prices are numerous, but Android offers less bureaucracy. Development time increases if cross-platform. Flutter or React Native would be necessary for cross-platform support. |
| **Development Tools** | Languages like Objective-C and more recently Swift for native apps, and JavaScript for web apps. Xcode is the primary IDE. Licensing costs for technologies can be substantial. Multiple development teams would not be necessary. | Languages like Python and C are standard for Linux. Any IDE is applicable. Git for version control as well. Little to no licensing costs. Multiple development teams would be preferred. | Languages like C#, C++ are standardized for Windows, mainly through the .NET framework, using Visual Studio as the IDE. SDKs simplify package management. Some licensing costs would be incurred. Multiple development teams would not be necessary. | Languages and technologies like Swift/Xcode for iOS, Kotlin/Java/Android Studio for Android, or Flutter for cross-platform. Requires mobile device emulators, other technologies for backend or CI/CD tools. Not much in terms on licensing costs. Multiple development teams would be needed. |

## 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**: Considering the multi-platform requirements of “Draw It or Lose It”, I am confident that a web-based approach is the most accommodating operating platform. Using HTML5 and Javascript frameworks like React would be ideal, as this choice supports cross-platform compatibility, ensuring a seamless experience across desktop and mobile devices.
2. **Operating Systems Architectures**: The chosen web-based architecture will be primarily client-server based. The client side of the application will be responsible for rendering the UI and handling user interactions, while the server-side will manage game logic, player data, and interactions between different instances.
3. **Storage Management**: For storage, a combination of cloud-based storage solutions like Amazon S3 for static assets and a NoSQL database like MongoDB for dynamic data would be effective. This setup offers scalability and flexibility, essential for a growing and dynamic game environment.
4. **Memory Management**: As a web-based environment, memory management largely depends on the client’s browser. Efficient coding practices, such as minimizing memory leaks in Javascript and optimizing asset loading will ensure smooth performance.
5. **Distributed Systems and Networks**: Utilize WebSockets for real-time, bidirectional communication between clients and the server. This will facilitate swift data transfer that is essential for a multi-player online game. This ensures timely updates and synchronization across multiple platforms.
6. **Security**: We would implement robust security measures like HTTPS for secure communication, OAuth for user authentication, and regular security audits to ensure data privacy compliance. Also, implementing features like data encryption and secure API endpoints to safeguard against common web vulnerabilities.