

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 | 02/15/24 | Justin Burkhalter | Initial software design document draft for The Gaming Room's web-based game application. |

## 

## [Executive Summary](#_sbfa50wo7nsh)

The software design problem entails transitioning "Draw It or Lose It" from an Android-exclusive application to a versatile web-based platform suitable for various devices. The proposed solution is a scalable, multi-platform web application that maintains the game's core mechanics while introducing enhanced accessibility and a new user interface adapted to web standards. Critical to our design is the unique identification of games, teams, and players, ensuring coherent single-instance gameplay across multiple sessions.

## Requirements

* **Team Structure:** May need separate teams for server and client-side development, especially to handle the variety of client platforms (iOS, Android, desktop browsers).
* **Expertise:** Requires developers proficient in cross-platform development and familiar with the latest web technologies.
* **Licensing Costs:** Mostly related to development environments on Windows and potentially for some Mac tools.

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

## [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 Domain Model for "Draw It or Lose It - Web Edition" includes Entity, Game, Team, and Player classes. Entity is an abstract class from which Game, Team, and Player inherit, encapsulating common attributes like id and name. The GameService class employs the Singleton pattern to ensure a single instance handles game management, while the ProgramDriver class acts as the entry point to the application. Using inheritance and encapsulation allows for an organized hierarchy of classes, each with distinct responsibilities.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Characteristics:** Known for stability and security, integrates well with other Apple products. **Advantages:** High-quality server hardware, less prone to viruses. **Weaknesses:** Higher cost, less commonly used in server environments compared to Linux and Windows. | **Characteristics:** Open-source, highly customizable, widely supported by cloud service providers. **Advantages:** No licensing fees, robust community support, stable and secure for web servers. **Weaknesses:** Requires more technical expertise to manage compared to Windows. | **Characteristics:** User-friendly interface, widely used in business environments. **Advantages:** Easy to set up and manage, extensive support for .NET applications. **Weaknesses:** Licensing costs, more vulnerable to malware compared to Linux. | Not typically used as servers due to hardware limitations and network constraints. |
| **Client Side** | **Characteristics:** Known for its high-resolution displays and advanced graphics, Mac provides a premium environment for web applications. **Advantages:** Ability to leverage advanced hardware for enhanced visual quality; strong integration with Safari and other browsers. **Weaknesses:** High user expectations for design and usability standards may require additional development efforts to meet the Mac user base's demands. | **Characteristics:** Linux supports a wide range of open-source web browsers and offers extensive customization options for developers. **Advantages:** Flexibility and control in the development environment, which can be tailored for performance and security. **Weaknesses:** Ensuring uniform functionality and appearance across various Linux distributions can be challenging. | **Characteristics:** With its broad user base, Windows supports a wide variety of devices, from desktops to tablets. **Advantages:** Large potential audience; compatibility with a wide range of web browsers. **Weaknesses:** The diversity of hardware and Windows versions necessitates extensive testing to ensure consistent user experience. | **Characteristics:** Touch interfaces and smaller screen sizes define mobile platforms, requiring a mobile-first design approach. **Advantages:** The ability to reach users on-the-go; optimization for touch interactions and mobile usability. **Weaknesses:** Hardware limitations and the need for performance optimization can complicate development; extensive testing across different devices and OS versions is necessary. |
| **Development Tools** | **Characteristics:** Offers a unified development environment, particularly through Xcode, which is tailored for developing applications on Apple's platforms. **Advantages:** High integration with Apple hardware and software, facilitating the development of optimized applications. **Weaknesses:** Some development tools and software may incur costs, and the ecosystem is more closed compared to Linux. | **Characteristics:** Embraces open-source development tools and environments, providing a wide range of options for developers. **Advantages:** No licensing costs for most tools, which can significantly reduce development expenses. **Weaknesses:** The diversity of tools and environments can lead to compatibility issues or require additional time for developers to adapt. | **Characteristics:** Dominated by Microsoft's development tools, such as Visual Studio, providing a comprehensive environment for .NET and other Windows-based application development. **Advantages:** Visual Studio offers a powerful, integrated development environment with extensive support for various programming languages and frameworks. **Weaknesses:** Licensing costs for Visual Studio and other Microsoft development tools can be a factor for budget considerations. | **Characteristics:** Development requires platform-specific IDEs like Android Studio for Android and Xcode for iOS, each supporting their respective programming languages and frameworks. **Advantages:** These IDEs offer extensive support for mobile development, including emulators for testing and debugging. **Weaknesses:** Requires developers to learn platform-specific languages and frameworks, which can increase development time and costs. |

## 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 Recommendation:** A cloud-based platform such as AWS or Azure is recommended. These platforms offer extensive support for various operating systems, scalability, and reliability essential for gaming environments.
2. **Operating Systems Architectures:** Both AWS and Azure support a variety of operating system architectures, including Windows Server, Linux distributions, and more. These platforms enable microservices architecture, allowing separate parts of the game to run independently and interact through well-defined interfaces.
3. **Storage Management:** Using distributed databases like Cassandra or MongoDB on these platforms can provide the scalability and performance needed for "Draw It or Lose It". These databases offer high availability and automatic scaling to handle varying loads.
4. **Memory Management:** Cloud platforms employ automated scaling and memory management, adjusting resources as needed to meet demand. This ensures efficient use of memory and improves the game's performance without manual intervention.
5. **Distributed Systems and Networks:** Communication between various platforms can be achieved through RESTful APIs or WebSockets, which support real-time data exchange for multiplayer gaming. This setup supports a distributed system where game instances can interact seamlessly across different devices and networks.
6. **Security:** Implementing SSL/TLS for data transmission and OAuth for authentication ensures the security of user information. These measures protect data in transit and at rest, safeguarding against unauthorized access and ensuring privacy compliance.