

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 | 08/15/2024 | Kashyab Maharjan | Initial version of the document. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is developing a web-based application to expand the reach of its current Android game "Draw It or Lose It" across various platforms. The program entails teams competing to deduce drawings from a collection of images at scheduled intervals. The primary objective is to create a web-based version that is capable of expanding and enduring, accommodating numerous teams and players. It must also enforce distinct game and team names and guarantee the existence of just one game instance at any given moment. This document provides a detailed description of the software design that will fulfill the specified criteria. The document encompasses details about the system architecture, outlines any constraints or limitations on the design, and recommends platforms for development.

## Requirements

1. Multi-Team Support: The game should be able to handle the presence of many teams, each consisting of numerous players.

2. Distinct Names: Within the system, game and team names must be exclusive.

3. Exclusive Instance: The game can only have one occurrence at a time, ensuring that distinct identities are assigned to each game, team, and participant.

## [Design Constraints](#_2et92p0)

1. Web-Based Environment: We must optimize the program to operate efficiently in a web-based setting. This entails ensuring interoperability across various browsers and devices, as well as optimizing performance to accommodate several concurrent users.

2. Scalability: The design should consider scalability, guaranteeing that the application can manage a higher workload as the number of users increases.

3. Data Integrity: We must implement measures to guarantee the uniqueness of game and team names, and to ensure that only one game instance is active at any given moment.

**Implications:**

* The application must employ efficient methods for storing and retrieving data in order to ensure optimal performance and consistency.
* Security measures must protect user data and prevent unauthorized access or tampering.

## [System Architecture View](#_ilbxbyevv6b6)

While the project does not require precise details in this section, it is crucial to understand the overall architecture. The system will comprise:

* Frontend:Web browsers can access the frontend, which is a user interface.
* Backend: This is the part of the system that deals with the server-side logic for managing the game. This includes managing user interactions as well as the game's current state.
* Database: A repository for storing data related to games, teams, and players.

An elaborate architectural diagram will depict the interaction among these components, encompassing communication protocols and data flow.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram has the following essential classes:

* Entity: A fundamental class that shares characteristics like `id` and `name`.
* Game:The `Game` class is derived from the `Entity` class and is responsible for overseeing the game's state and handling interactions.
* Team: A subclass of `Entity` that represents a team in the game.
* Player: A class that extends the `Entity` class and represents individual players.

**Object-Oriented Principles**

* **Inheritance**: The `Game`, `Team`, and `Player` classes derive from the `Entity` class, enabling code reuse and maintaining consistent properties.
* **Encapsulation**: Each class encapsulates its data and provides methods for manipulating it, ensuring a distinct division of 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** | The lack of widespread use of Mac servers for web hosting can lead to limited support and resources. The overall performance is satisfactory, but the ability to handle huge loads may be a potential issue in terms of scalability. | Linux is exceptionally effective for web hosting, exhibiting robust speed and scalability. It enjoys extensive usage and support. | Although Windows provides robust support and a diverse range of development tools, it may entail license expenses and performance challenges when subjected to severe workloads. | Mobile devices are typically not suitable for hosting; however, responsive web designs can guarantee compatibility. |
| **Client Side** | Developing for Mac clients may incur more expenses due to a smaller pool of professionals and higher hardware rates. Proficiency in macOS-specific development is necessary. | Specialized development tools and abilities may be necessary for Linux clients. With a lower market share, there is a possibility of a smaller user base. | Windows is the predominant client platform, providing extensive support and a wide range of programming tools. The widespread use of a certain technology or practice allows for control over both cost and time. | Supporting mobile devices necessitates distinct considerations for user interface and user experience, as well as extra development effort to ensure responsiveness |
| **Development Tools** | The languages and technologies encompass Swift and Xcode, as well as other cross-platform frameworks such as Electron. | The languages and tools encompass Python and Java, as well as a range of integrated development environments (IDEs) such as Eclipse or VSCode and frameworks like Node.js. | The available languages and technologies encompass.NET, Visual Studio, as well as an extensive array of frameworks and libraries. | Development entails utilizing frameworks such as React Native or Flutter to provide cross-platform interoperability. |

## 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:** After careful assessment, Linux is the recommended choice for server-side hosting due to its high efficiency, scalability, and extensive use. When it comes to client-side development, it is important to prioritize Windows because of its extensive user base. To expand your reach, you can use cross-platform frameworks to achieve mobile support.

2. **Operating Systems Architectures**: For the server-side architecture, the selected operating system, Linux, will employ a LAMP stack (Linux, Apache, MySQL, PHP). Windows clients will utilize web technologies that are compatible with widely used browsers, while mobile devices will employ responsive design techniques.

3. **Storage Management:** Employ a relational database management system (RDBMS) such as MySQL or PostgreSQL to manage game, team, and player data. Verify that the database schema includes unique restrictions for names and enables efficient querying.

4. **Memory Management**: Linux employs advanced memory management algorithms, ensuring efficient process administration and memory allocation. Implementing caching mechanisms and optimizing queries can significantly improve performance.

5. **Distributed Systems and Networks**: The application will employ a distributed system architecture, utilizing RESTful APIs for inter-component communication. Deploy load balancing and failover techniques to manage network problems and service disruptions.

6. **Security**: Employ SSL/TLS to establish a secure connection between the client and server. Employ encryption to safeguard sensitive data and establish strong authentication and permission processes. Regularly update and patch the system to prevent vulnerabilities