

Draw it, or loose it!

# **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 |
| --- | --- | --- | --- |
| 1.2 | 04/21/2024 | Alexander DeMarco | Created entity class for game package, updated evaluations for further development. Added recommendations to preferred architecture. |

## [Executive Summary](#_sbfa50wo7nsh)

“The Gaming Room” tasks Creative Technology Solutions to provide a web-based game that is accessible for multiple platforms for their current game “Draw it, or Lose it”, which is only available in an Android app only.

## Requirements

Game to have ability to have one or more teams involved. Each time will have multiple players assigned to it. Each game and team must have unique names for each team and must pass a check before proceeding. Only one game instance can exist at any given time, so unique identifiers must exist for each instance.

## [Design Constraints](#_2et92p0)

1. **Playing together online**: Making sure that when playing together on the web, there it is real-time.
2. **Works on all devices:** Whether using an old computer, new mobile device, or tablet, all should be viewing the game on all devices that looks sharp.
3. **Keeping it unique:** Team names should be unique, meaning no one else can use it. Will need a smart way to check to see if name is already taken.

## [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)

**Entity:** This is the basic building block for the game. All teams, players, and the game start here, providing a unique identifier for each.

**Game Class:** This is the whole game setup. This will include everything needed to play and which teams are playing.

**Team Class:** This class tracks the team’s name and the player assigned to the team.

**Player Class:** Each person who is playing the game. This will include the individuals’ names and what time they are on.

**Connections:**

The game has teams, whereas players are assigned to a particular team. This start is the basic building block (entity)

Teams and players have unique names. This is like sports teams, where a player is wearing a team jersey with a unique number, where no other team member shares with another except for the team’s name.

**Inheritance:** The game, teams and players all share basic things. This nestles the ID’s and names and reduces the overall code.

**Organization:** Overall, the game is organized so that it knows which teams are playing and who is on each time. This ensures that the game will run smoothly.

**"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 must 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** | Unix-based  Excells at web hosting  Limited scalability  Flexible terminal commands to configure server, access or make changes.  Are upgradeable, various options for web hosting. Server cost is inexpensive.  Less preferred for web hosting. | Open-source system, widely used for web hosting for high stability, security & scalability.  Compatibility issues with other software  Same as Mac, but more cost friendly.  More secure and is preferred for web hosting.  More difficult to find support for an application.  Many users not familiar with Linux | Familiar environment for developers, seamless integrations with wide range of software.  Limited support for open-source web development.  High resource requirement, but less loading times and high comfort.  Susceptible to viruses  Can be costly but is the most used system – reaching a wide range of people | iOS and Android systems provide diverse eco systems.  Mobile platforms give the user enhanced accessibility.  Better to have server as immobile.  More popular and highly portable.  Highly selective to devices and poor security  Mobile devices are not designed to host servers as they have limited capability in that regard. |
| **Client Side** | Cost: require investment in Mac hardware to test development.  Time: requires more time for development  Expertise: macOS knowledge required  A less used system compared to Windows. | Cost: Linux is distributed for free, so lower costs are involved  Time: Testing will be time-consuming. Linux can be difficult for even experienced developers.  Expertise: knowledge of Linux tools is necessary. Minimum cost to ensure compatibility. | Cost: Licensing costs may be high  Time: Ensuring compatibility for various window versions may increase time, but experienced developers could prove fruitful.  Expertise: knowledge of VS.  Cost is like MAC. | Cost: increased due to multiple development tools and device testing for platform.  Time: Increases testing time to optimize platforms. Mobile devices are easier to work with, but there are many devices distributed and may need extended time to test variety of them  Expertise: proficiency in mobile development frameworks.  Provides flexibility to clients and developers for updates.  Difficulty to implement due to variety of devices. |
| **Development Tools** | JavaScript, Objective-C. macOS specific IDE would be Xcode.  Homebrew & Carthage can be utilized as well | Python, JavaScript  IDEs include VS code, IntelliJ & Eclipse.  Git, Docker, Bash can be integrated as well. | C#, JavaScript, Python.  IDEs include VS, Eclipse, PyCharm.  .NET framework may be needed. | Swift (iOS), Kotlin (Android), JavaScript  IDEs include Xcode, Android studios, Visual studio.  Android SDK, iOS SDK, React Native |

## 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**: Linux web hosting would be the recommended operating platform. It is cost effective, provides scalability & security. It is also highly compatible with various software. Despite the learning curve for developers, it has a vast community and resources available for support.
2. **Operating Systems Architectures**: Linux is well suited for general purpose computing for a game such as this. Linux supports web hosting servers and the game’s backend computing.
3. **Storage Management**: Recommendation of using a Relational database such as MySQL. This database is well suited to handling game entities such as player teams and game instances. It can also handle images if the Game Room wishes to fetch images, considering memory management.
4. **Memory Management**: It is recommended to fetch from the storage database and store in the virtual memory only when needed. With caching, more frequently used players and images will speed up response. This will reduce the disk I/O operation for file reading/writing.
5. **Distributed Systems and Networks**: It is recommended to use a service orientated principle and RESTful API’s. The game functions would be divided into independent services. This will ensure cross-platform compatibility – which will provide consistent user experience. Additionally, a network infrastructure with redundancy will minimize connectivity issues to ensure seamless experience for the game player.
6. **Security**: Multi-factor authentication OAuth would ensure only authorized users can access the game features and data. It is also recommended that sensitive user data is encrypted for both data at rest and in transit. Implementing role-based access ensures users only access functions relevant to their role, preventing authorized users from gaining access to administrative functions. Using HTTPS protocols ensures that data exchanged between platforms is encrypted to protect from eavesdropping or tampering.