

Draw It of Lose 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 |
| --- | --- | --- | --- |
| 1.0 | 02/01/2024 | Austin Bland | Added Executive Summary, Design Constraints, and Domain Model. |
| 1.1 | 2/13/2024 | Austin Bland | Adds the evaluation. |
| 1.2 | 03/01/2024 | Austin Bland | Adds recommendations. |

**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)

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* 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.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* 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](#_2et92p0)

* Network Latency and Bandwidth: network requests need to be minimized; game data transfer needs to be optimized.
* Scalability: needs to be able to handle fluctuations in player numbers and server loads.
* Security: Player data needs to be secure. Cheating may also need to be prevented.
* Cross-Platform Compatibility: Needs work on different devices, browsers, and operating systems.
* Data Consistency: Players may all have different browsers and operating systems driving the need for everyone’s game experience to be the same without bias toward any OS or browser.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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.**

ProgramDriver uses the SingletonTester class to ensure that only one ProgramDriver is in use.

Entity class is the parent or super class of Game, Team, and Player classes. It contains methods that Game, Team and Player will all use to keep the code clean and efficient, thus eliminating redundant methods within the child classes.

GameService class contains the methods that will be used in the ProgramDriver that will perform all the tasks related to the games functions.

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | User Friendly Interfaces  Uses macOS Server  Limited Scalability  Higher Cost | Stable  Secure  Scalable  Cost Effective  Less Commercial Support | -Often Used for Enterprise  -Well Supported  -Integrates with Microsoft Products  -Less Secure with Higher Cost | -Large User Base  -Limited Capabilities compared to desktop or server platforms.  -May require additional testing efforts |
| **Client Side** | More expensive for Hardware and Licensing.  Mac Tools are not the most widely used. Developers will need time to familiarize themselves with them.  Need to be effective at macOS specific technologies such as Swift and Cocoa frameworks. | More affordable for both hardware and software is free and open source.  May have a learning curve but is well supported.  Proficiency in Linux development tools will be needed. | Wide range of Hardware at a wide range of costs.  Windows based platforms are widely used but may require learning for specific applications.  Working knowledge with Visual Studio and the .NET framework are essential. | Hardware costs vary by brand. Development tools will require licensing that is not free.  Development time could be increased due to design and testing for mobile devices.  Expertise in platform specific development tools is necessary. |
| **Development Tools** | macOS is Unix based and supports various programming languages and development tools.  Xcode IDE and Interface Builder are used often.  Cocoa frameworks work for macOS and IOS development.  Poor support for non-Apple products. | Linux widely supports various programming languages and development tools including open-source options.  Advantages are the use of the GNU Compiler Collection that is used in C programming.  User interface may be less polished. | Windows supports several development tools and languages.  Advantages include the Visual Studio IDE and the use of .NET framework that are exclusive to Windows.  Limited flexibility and is not open source. Can come with a learning curve for some. | Mobile Development requires platform-specific tools and languages for IOS and Android.  Xcode IDE is good for IOS and macOS development.  Andriod Studio is the IDE for developing Andriod applications.  Cross platform frameworks are available. |

## 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**: I recommend the use of a Linux Web Based operating platform for its security, ease of use and scalability.
2. **Operating Systems Architectures**: The clients will access the game through their web browser. The browser will send requests to the Linux server. The server then hosts the game application for the duration of use.
3. **Storage Management**: Game data will need to be stored for the duration of the game. With a Linux server with the use of a MySQL relational database on the server will be a good choice to store all the game service data.
4. **Memory Management**: The Linux server side will allocate memory to handle incoming requests and execute the game logic. Memory will be dynamically adjusted to best suit the gaming experience. Which included de-allocating memory as needed to keep the game efficient. For the user who’s computer must render the game, we will need to ensure that we will appropriately utilize processing to keep the game running clearly while still keeping it simple enough to run on most computers.
5. **Distributed Systems and Networks**: Since the game needs to run on various platforms, a distributed software architecture can be implemented. The distribution will need to specifically be tailored for this game’s specific requirements. The distribution will need to utilize communication protocols that ensure the connection between the client and server. It will need to break the game service into smaller services and containers to simplify the games use on different platforms. Additionally, it will need to utilize frameworks, libraries and languages that support cross compatibility for different platforms.

By adopting a distributed software architecture and considering network dependencies, Draw It or Lose It can achieve seamless communication between various platforms, providing players with a unified gaming experience while mitigating the impact of connectivity issues and outages.

1. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>Security needs to be robust. I think that a Secure Socket Layer or SSL protocol for communications between the client and the server. This ensures that the data being exchanged during gameplay stays secure. Next, we need to implement Role Based Access Control or RBAC. This will ensure that players and other users can only access what they need and cannot use the game at an Admin level. This keeps the deeper software more secure. I additionally recommend that we implement a form of two factor authentication to validate that the players that sign in are who we expect. Lastly, we should have a system in place that is constantly searching for threats. This will mitigate the chances that if someone does get in, they won’t be there for long.