

Draw It or 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 | 03/19/23 | Benson Hunt | Initial 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 software design problem is to create a web-based application that allows teams to compete against each other in a game of guessing phrases or titles represented by stock drawings. The client has provided specific software requirements, including the ability to have multiple teams, each with multiple players assigned to it, unique game and team names, and the ability to ensure only one instance of the game can exist in memory at any given time. To meet these requirements, we propose developing the application using the Model-View-Controller (MVC) architecture, with six components: Game Controller, Team Controller, Player Controller, Puzzle Generator, Drawing Renderer, and User Interface. The application will generate unique identifiers for each instance of a game, team, or player, to manage the state of the application and prevent conflicts between different instances. This solution will meet their requirements and provide a robust, scalable, and maintainable application that can handle multiple users and teams playing the game simultaneously.

## Requirements

1. Each team will have multiple players assigned to it.
2. Ability to have one or more teams involved in the game.
3. Unique game and team names to allow users to check whether a name is in use when choosing a team name.
4. 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)

1. Performance: The application must be fast and responsive, even when handling multiple teams and players at once, to ensure that users can enjoy the game without experiencing lag or delays.
2. Scalability: The application should be designed to handle a growing number of users and teams as the popularity of the game grows.
3. Security: The application must be secure and protect user data and information, especially when handling login credentials and personal details.
4. Compatibility: The application must be compatible with a wide range of web browsers and devices to ensure that the game can be played by the largest possible audience.
5. Usability: The application should be easy to use and navigate, with clear instructions and user-friendly interfaces, to ensure that users can quickly learn and enjoy the game.
6. Cost: The development and maintenance costs of the application should be reasonable and within the client's budget.

## [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 UML below uses ProgramDriver for a starting point of the Application. The ProgramDriver uses the SingletonTester. The Game, Team, and Player all inherit from entity. The GameService can have 0 to an infinite about of games at a time. The GameService is A singleton meaning it will only be created once. The Game can have 0 to an infinite number of teams. Teams can have 0 to an infinite number of players.

**"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** | User-friendly interface: User friendly to host on  Compatibility: Works well with other apple products  High-Quality Hardware: High quality hardware means less failures in the future.  Price: Very High Compared to other options  Limited Software Options: Software is made normally for Windows or Linux and not as often for mac. | Cost: free to use  Customization: near limitless  Reliability: can run extremely long periods without downtime  Complexity: Very hard for novice users to use  Limited Software Support: Many tools and apps are made for Windows and Mac but not Linux. | User-friendly: user friendly to host on.  Compatibility: Most software is made for Windows.  Cost: Normally has licensing fees required for commercial use  Security: Windows is often targeted for security vulnerabilities. | Convenience: Everyone has them  Limited Processing Power: only so much processing can be leveraged meaning web-based application can suffer on performance.  Battery Life: Doesn’t have a direct line to the power grid unless always plugged in.  Connectivity: Unreliable connection with the internet. |
| **Client Side** | Web-based Apps would require no extra time or expertise to work with Mac. | Web-based Apps would require no extra time or expertise to work with Linux. | Web-based Apps would require no extra time or expertise to work with Windows. | Require a minimal extra amount of time to get screen size and interface to work together. |
| **Development Tools** | Eclipse IDE  SDK | Eclipse IDE  SDK | Eclipse IDE  SDK | Not recommended |

## 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**: Windows as it cost less than Mac but gives an easy to understand and use way of hosting the software.
2. **Operating Systems Architectures**: x64 is the most common and is supported by most software.
3. **Storage Management**: Relational databases would be the most maintainable and easiest to work with.
4. **Memory Management**: Using virtual memory allows applications to use more memory than present in the system by swapping data between RAM and disk.
5. **Distributed Systems and Networks**: To implement the distributed system architecture, the Draw It or Lose It software can be divided into multiple components, each running on different platforms, such as desktops, mobile devices, and servers. These components can communicate with each other over the network, allowing the different platforms to work together seamlessly.
6. **Security**: Installing built in security such as firewalls and anti-virus along with encryption to secure network traffic would protect user information from unauthorized access.