

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 | David France | Updated to include Executive Summary, Requirements, Design Constraints, and Domain Model. |
| 1.1 | 04/01/23 | David France | Updated Evaluation section to include evaluation of different operating systems. |
| 1.2 | 04/14/24 | David France | Completed all areas of Recommendations section. |

**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 Gaming Room would like to expand their Android based game, Draw It or Lose It, to become a multi-platform web-based game. They require an environment to be set up that can be adapted to different platforms. That environment needs to ensure that only one instance of a given game can exist in memory at a time. It also needs to ensure any game can be made up of one or more unique teams, with each team consisting of multiple players.

## Requirements

* Each game can have one or more teams.
* Each team has multiple players.
* Each game and team must be unique.
* Only one instance of any game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

* Programming language will be Java.
* Environment will be made up of multiple classes – Game, Team, and Player classes will all inherit from parent Entity class.
* A singleton design pattern will be used to ensure only one instance of each Game can exist in memory.
* An iterator design pattern will be used to ensure game and team names are unique.
* No budget or schedule has been agreed to yet, this will need to be addressed.

## [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)

In the UML class diagram, classes Player, Team, and Game all inherit from Entity. This ensures they all have a name and ID, along with getName() and getID() without duplicating code. Among those three classes, one Game is made of many Teams, and one Team is made of many Players. The GameService class then can be made of many games, but ensures only one instance of each individual Game is allowed at a time. GameService is a singleton, so no duplicate GameService instances can be created. GameService also holds the values for nextGameId, nextTeamId, and nextPlayerId so there are no duplicates.

As discussed, inheritance is shown by Game, Team, and Player classes inheriting from Entity. This creates an efficiency by preventing duplicate code. Separating the project into classes exhibits portability, making it easy to identify where problems may be and to use those classes in other applications. The singleton pattern is an example of encapsulation, which ensures only one instance of each game exists in memory. Encapsulation is also exhibited by each class hiding its data components. Polymorphism is shown by multiple objects, i.e. Entity and Player, can be created with a common interface.

The ProgramDriver class runs the Main() program and the SingletonTester class is used to check that only one instance of GameService exists.

**"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** | Macs can be used for hosting a web-based software application, but they are a distant third behind Linux and Windows. While it is more secure than Windows, it’s also much more expensive than Linux. The main argument for using Mac seems to be if it’s the OS you’re already comfortable with or you’re trying to develop applications for Mac or iOS. | Linux is the most popular OS for hosting web servers. It offers all the desirable qualities of stability, security, and affordability. The down side is it can be difficult to use by comparison to others, so you need a team that can work with it. Given the upside, ensuring our team had developers who can work with Linux should be a priority. | Windows is the second most popular OS for hosting web-servers. It’s easier to use than Linux, and familiar to most users. The downsides are it’s more expensive than Linux, while also being the most vulnerable of the three to security threats such as malware. | Hosting a web-based software application can be resource intensive and beyond the scope of most mobile devices. |
| **Client Side**  **\*** We are using a client-server design pattern. This allows us to take HTML input from all three through our web-based game. This means the client side considerations are the same for all three desktop operating systems. If we needed to develop a game for each system it would require a large increase in time and money, as well as ensuring we had developers for each OS. | The development of a web-based game means having an HTML interface within the web browser for Mac clients. The client can interact with our site, which takes HTML input from all clients. From a development perspective, we would need to develop the game in Java, the website in HTML, then create a RESTful API. These put together would allow us to handle the needs of Mac users. We would need both Java and HTML developers. The cost and timeline will be the same for all three desktop operating systems, as we’re developing a product to handle all three. | The development of a web-based game means having an HTML interface within the web browser for Linux clients. The client can interact with our site, which takes HTML input from all clients. From a development perspective, we would need to develop the game in Java, the website in HTML, then create a RESTful API. These put together would allow us to handle the needs of Mac users. We would need both Java and HTML developers. The cost and timeline will be the same for all three desktop operating systems, as we’re developing a product to handle all three. | The development of a web-based game means having an HTML interface within the web browser for Windows clients. The client can interact with our site, which takes HTML input from all clients. From a development perspective, we would need to develop the game in Java, the website in HTML, then create a RESTful API. These put together would allow us to handle the needs of Mac users. We would need both Java and HTML developers. The cost and timeline will be the same for all three desktop operating systems, as we’re developing a product to handle all three. | The development for mobile devices faces the same considerations as for Mac, Linux, and Windows, with the additional requirement that the website be mobile-friendly. This would increase the time, and cost of developing the site. |
| **Development Tools** | Swift or Objective-C are common programming languages for Mac. Xcode is an IDE that supports both languages. | Linux supports most major languages, from C++ to PHP, Perl, Ruby, Python, or Java. As Java is so prevalent for web-based software, that would be the likely choice. Eclipse would be the IDE to use in this scenario. | Windows supports many languages, including C++, PHP, and Python. Java is the most popular language for developing web-based development, and has been used for games such as Minecraft. Eclipse is a popular IDE to develop Java code. | Java or Kotlin with Android Studio for Android and Swift and Xcode for iOS. |

## 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**:

We recommend that The Gaming Room operates using web-based services. These include a serverless architecture for memory management and the cloud for storage. Should they want to operate their own server, we recommend that The Gaming Room uses Linux as its operating platform. It offers stability, affordability, and security compared to Windows or Mac.

1. **Operating Systems Architectures**:

Going with a web based service means a serverless architecture for The Gaming Room. This can take the form of microtransactions for each activity they need – running the game, taking user information, running ads. It also means interacting with a cloud storage company for handling the storage of image files and user information.

1. **Storage Management**:

The Gaming Room needs to be able to store a large library of images, as well as historical game data, including images of drawings. Given that, we recommend they leverage the cloud to host their storage needs. Cloud storage is a close to perfect efficiency as you can get, while being able to handle vast amounts of data. With this model, you only pay for the storage you use, so there’s no wasted capital. It is also possible to scale very quickly with a growing business.

1. **Memory Management**:

The gaming room should use a serverless architecture to handle memory management. This is as opposed to investing capital to host their own memory. Serverless architecture allows user to pay by the microsecond or transaction instead of physical infrastructure. This is both an efficient use of capital and would allow The Gaming Room to expand capacity rapidly. They could also easily add new environments, such as other games or more features to Draw It or Lose It. As a further benefit, using a serverless architecture reduces the need for backend support, allowing The Gaming Room to focus on their core business.

1. **Distributed Systems and Networks**:

The client-server pattern is an efficient solution to running a web-based application on multiple operating platforms. Basically, the users interact with a front-end server, then that front-end server interacts with the software/database on the back end. A common analogy is a server in a restaurant taking an order from many diners, then taking those orders to the kitchen. Using a REST API style of communication allows the input of HTML from any user and outputs JSON to the backend server. In this way, The Gaming Room does not have to worry about creating different versions of the game for each platform.

1. **Security**:

The Gaming Room will need a way of protecting user information. They will be using cloud services for storage and memory, so physical security will be the concern of the companies running those solutions. That leaves customer information security and the human element within the company up to the Gaming Room.

As far as the human element is concerned, The Gaming Room’s employees will have access to the backend of the operation. This leaves a large potential risk if someone wanted to get into the network through The Gaming Room or one of its employees. To account for this, employees need to be trained on common phishing scams, password security, and overall network security.

For username and password security as far as customers are concerned, all usernames and passwords will need to be encrypted. Even if someone is able to access the database, they should not be able to read the information. On top of this, two factor authentication has become the standard for password security. Even though The Gaming room is just running a simple game over a web service, they will still be handling sensitive customer information and need to take that seriously.