

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 | 11/11/2023 | Kyle Cortez | -Created Base Class Entity for Games, Teams, and Players to inherit from.  -Ensured only one instance of any Entity can exist with the same name and ID in memory. |

**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)

We need to ensure that the porting of this game to a web based version will need to have the ability to link the app with a database using something like SQL or Mongo(for Non SQL). In addition, the requirement for single instance objects for games rooms, teams, and players with unique IDs will be imperative to ensure the integrity of the database. With web based, the importance of secure code cannot be understated as to curb cheaters.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

-File and image sizes must be compressed and optimized for slower connections.

-Data collection may be necessary to ensure security but necessary security precautions are needed to ensure integrity.

-Integrity of data to and from client and server is a must. Standard encryptions are needed to ensure secure connections and communications to prevent bad actor circumventions.

## [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 classes Game, Team, and Player classes all derive commonly used data layouts from the Entity class through Inheritance. Since Entity should never be instantiated, we can Abstract this class.

Since there will only be one instance of Game service, a singleton pattern is necessary and points back to the created instance for the server.

There can be many Game, Team, and Player instances, however, each instance must be unique to ensure data integrity for use in a database.

**"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** | -Higher security with IOS and tighter security on source code.  -Specialized developers so may cost more per man hour.  -IOS development may cost more to use software and tools | -Relatively secure due to open source.  -Less ubiquitous lowering compatibility.  -Learning curve for application development.  -Cost effective tools and software with open-source licenses. | -Cost effective with high developer base.  -Ubiquitous with a lower learning curve for tools and software.  -Quick development cycles with well-established standardizations. | -Outsourced to cloud services such as Azure or remote server running Windows, Linux, or Mac.  -Costs can vary depending on server provider. |
| **Client Side** | -Higher security with IOS and tighter security on source code.  -Specialized developers so may cost more per man hour.  -IOS development may cost more to use software and tools | -Relatively secure due to open source.  -Less ubiquitous lowering compatibility.  -Learning curve for application development.  -Cost effective tools and software with open-source licenses. | -Cost effective with high developer base.  -Ubiquitous with a lower learning curve for tools and software.  -Quick development cycles with well-established standardizations. | -Higher dev time for optimizations to ensure compatibility with older systems.  -Extra dev time for cross platform compatibility. |
| **Development Tools** | Xcode with CocoaPods for dependencies | -Java tools such as IntelliJ or Eclipse with Docker | -Java tools such as IntelliJ or Eclipse with Docker | -Java tools such as IntelliJ or Eclipse with Docker |

## 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**: For Draw It or Lose It, recommendations are for serverless to allow better scalability and ease of development. Costs should be lower with this if no server base exists with client.
2. **Operating Systems Architectures**: Serverless should be able to be used to ease development time and costs associated. Client side should be written to accommodate this arrangement and thus should be scalable as the serverless host should be compatible with the multitude of devices desired by client and easy to port.
3. **Storage Management**: Recommendations for using cloud storage with a small local cache for access.
4. **Memory Management**: The use of cloud storage relieves much of the burden of disk space from the user thereby increasing both usability and scalability. A smaller local cache can be used to offset the need of higher wireless network speeds by preemptively downloading commonly used items. This combination serves to maximize users that can use the application while optimizing for lower end users.
5. **Distributed Systems and Networks**: By using distributed software, development can be focused on the client side after the initial legwork has been completed for the backend, thereby streamlining and lowering costs of development. The client side has much of the processing work relieved by just having a server-client architecture and with the results being displayed client side over network, this means that each client side, at minimum, needs only to receive and display information. While this is a boon, by the same token if a network goes down or server loses connection, there is nothing for the client side to receive and display. However, with the reliability of networks and connection being high, this is a profit in pros vs cons.
6. **Security**: The break between client and server in the software does introduce vulnerabilities but these can be protected against with the proper procedures and security protocols. First and foremost, concentration should be on database integrity and security. This includes login methods being airtight, for the case of something like SQL injection, and only storing or sending over network what is necessary for authentication and operation, for the case of interception. Encryption protocols should be adhered to for over network and storage of sensitive information and ensure the integrity of what is being captured on either end and validations to check that intake information.