

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 4.0

Victor Jarvis

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 + 4.0 | 06/2022 | Victor Jarvis | Fleshed out entire document with anymore necessary information. Some very light edits in mid/late June |
| 2.0 | 05/2022 | Victor Jarvis | Added platform evaluation and architecture recommendations, along with additional info. |
| 1.0 | 05/2022 | Victor Jarvis | Initial documentation, used to establish base information and summaries. |

**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 wants to develop a web-based game based on their current game, Draw It or Lose It, currently only available as an Android app. TGR would like to implement a version that serves multiple platforms. This document serves as a product summary and outline of the constraints that will affect the product, along with potential solutions.

## [Design Constraints](#_2et92p0)

The main constraints in designing the product center on the development team (including but not limited to anyone involved in any manner of methodology employed (ie, Scrum Master, stakeholder, etc)). The dev team will have to focus on UX/UI and easily maintainable features. As the current form of the app is Android based, it would be best to program the product in Java, which is Android’s official development language. Currently there are no known restrictions regarding software licensing and regulatory compliance. Any that arise during production should be handled respectively. At current the client does not have any constraints in regard to operating system, but given that the application is mainly web-based, there should be a focus on making the application compatible with all leading web browsers, such as Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft’s Edge.

## [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 Entity class is the base at which we form our foundation for the several subclasses beneath it, Game, Team, and Player, with each of these subclasses sharing attributes like name and an identifier. Going from left to right, the four classes along the bottom of our UML diagram represent associative relationships. For example, a Game object can consist of multiple Teams, which consist of multiple Players. Our GameService is meant to manage the Game that is being played, which there can only be one of at a time. Our singleton design, which restricts service creation to one instance, ensures this. The SingletonTester and ProgramDriver share a direct relationship, with the latter using the former to validate the services creation of a unique Game instance.

**"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** | High costs to be considered for licensing. Many features and software packages that have open source counterparts on other OS. Less technical constraints than Windows OS. | Open source which reduces technical constraints (lowest of the operating system choices), but security issues may be a problem due to file structure. Linux does have the best processing power | High costs to be considered for licensing. Many features and software packages that have open source counterparts on other OS | Mobile first development must be considered, but there are several frameworks that can be utilized and several types of dev tools that support various OS |
| **Client Side** | Does not support all file structures, so limits of media types presents a constraint that can be solved through additional programming work. RESTful structures support a need for processes that handle many requests quickly and efficiently. | Does not support all file structures, so limits of media types presents a constraint that can be solved through additional programming work. | Product must be optimized to run on Microsoft web browsers, though most should be supported. | Lightweight RESTful structures provide added efficiency as mobile processing power is too low for heavier loads and file types. |
| **Development Tools** | Multi-thread and one-to-one thread model. Eclipse IDE and open source frameworks (Maven, etc) provide deployment solutions. | Multi-thread and one-to-one thread model. | One-to-one thread model, limited technical constraints. | Different frameworks possibly for cross platform development, due to lack of support for Java on 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**: Linux OS, running on a cloud based or serverless architecture. Cloud based architecture allows for scalable and easily maintained solutions. Serverless pricing models that allow for a “pay for what you use” solution will also reduce cost. Use of an outside party for serverless architecture solutions greatly reduces complexity and allows for better support solutions.
2. **Operating Systems Architectures**: Three-tier server solutions and serverless architecture recommended. Middle layer solutions can improve the applications features and provide services for security, content handling, logic, and information handling. Linux’s lighter weight modular structure allows for less taking up of disk space.
3. **Storage Management**: Through serverless solutions, there is a great potential to increase efficiency, especially if combined with a more direct access storage method. This would allow for a better UX, as well as much faster load times. Indexed direct access storage would also get rid of wasted storage space. Linux’s file system structure offers this indexing though multiple tier allocation, allowing to conserve memory space.
4. **Memory Management**: Linux’s usage of virtual memory and demand paging allows for the elimination of physical memory constraints and improves the UX, as this also helps to load and run the application quicker. Ensuring the application’s code base is streamlined will also ensure better load times.
5. **Distributed Systems and Networks**: Employing serverless architecture will help to consistently scale the application as usage and the user base grows. Load balancing and synchronizing databases to seem as though they’re a single instance will help to spread requests evenly across servers. Utilizing an API to serve all clients will assist in supporting multiplatform usage of the application.
6. **Security**: User login credentials, (username or email and password) are recommended, with one-time password reset being a potential option if the user requires a reset. Users are assigned player roles that allow them to play the game but doesn’t allow them to make any changes with the application. Admin are assigned roles that allow them to manage files, puzzles, solutions, and any application configuration. Admins are also able to moderate and verify player user info.