

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 09/22/2024 | Tony Aguilar | Update executive summary, design constraints, domain model. |
| 2.0 | 10/06/2024 | Tony Aguilar | Update evaluation |
| 3.0 | 10/20/2024 | Tony Aguilar | Updated Recommendation |

**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)

Our client “The Gaming Room” has tasked us with expanding their current application, available on the android platform, “Draw it or Lose it” on the other available platforms such as Windows and Apple. The goal is to create a seamless experience for all users alike, but considering certain limitations, we will decide whether cloning the current user interface or creating a completely new one for the other platforms is better. We will set up the environment on which these versions of the application will operate and provide suggestions as needed.

## 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)

Functional Requirements

* Utilizing library of stock images
* Image rendering servers
* Database for validation of names
* Device requirements and limitations
* Cloning of current UI or fragmentation for other Operating Platforms

Non-functional Requirements

* Ability to run on popular operating systems such as Windows, Apple while continuing to run on Android.

- Ability to have one or more teams play at once

- Ability to have one or more players per teams

- Ability to validate that a team name is unique and not previously used

- Ability validate that a game name is unique and not previously used

- Ability to create unique identifiers for each game instance to prevent multiple instances from running

## [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)

Firstly we start with the system which is called “com.gamingroom” where all of our classes live. Secondly we have our class called “Entity” which is a parent class(superclass) of “Game” , “Team” and “Player” which makes them child classes”(subclasses). Some of the attributes in “Entity” are public therefore they are inherited by the child classes named above. This is also illustrated by the open arrow. This also means that “Entity” keeps our code “DRY”. We then have association between “Game” to Team with multiplicity of “0 to many” and between “Team” and “Player” again with multiplicity of “0 to many”. “GameService” holds methods for “Game”, “Team” and “Player” assuring that we keep the code “DRY” as well as validate the uniqueness of each of them. Lastly we have our “ProgramDriver” class with the main method that is tested by the “SingletonTester” subclass that tests for a single 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** | Its advantages are its walled-off garden that is very secure and usually developer friendly. Its downside is the higher cost as well as limitations to being able to implement across multiple platforms. | The advantage is that it is open source, so it is low cost and there is lots of opportunities for available tools. It is also very secure. The downside is that not every environment can host. | Widely available and most compatible amongst the different platforms. The groundwork is there, it just needs to work on implementations. The downside it is more vulnerable because of how popular the platform is. | The benefit is that it is lightweight and doesn’t require too much upkeep. The downside is that there are so many different devices to account for and to optimize that it becomes difficult to find a happy medium. |
| **Client Side** | Very user-friendly and easy to pick up. Seamless experience across multiple devices considering again the ecosystem. Very specific needs for development experience. | Given that it is a platform that is mostly niche and not as public as windows or mac, it is harder to have experts to work on this, so smaller talent pool and more difficulty in implementation. | Plenty of experts to develop and implement, so lower cost from that end, however the cost rise in order to be able to publish and maintain in this platform considering it is not open source. | The benefits are again that once developed, there is a seamless experience that can be easily updated and adjusted on the go. The downside is fragmentation across the different versions and devices. |
| **Development Tools** | IDE’s:  XCode, Visual Studio Code  Programming Languages:  Swift, Objective-C, Java, C++  Tools:  Homebrew,Dash,Raycast | IDE’s:  Visual Studio, Atom, Sublime  Programming Languages:  Python, C++, Java, Rust,  Tools:  Git, Blender, React Native | IDE’s:  Visual Studio, Eclipse, PyCharm, WebStorm  Programming Languages:  C#, Java, Python  Tools:  Docker, node, jest | IDE’s:  Android SDK, SwiftUI,Xcode  Programming Languages:  Ruby on Rails, Go, Swift, Objective-C  Tools:  Android Studio, Flutter, React Native, Ionic |

## 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**: The most developer and client friendly operating platform in this instance will have to be Windows due to its longstanding track record, usability and overall availability of talent as well as its affordability and usual package services. Considering that “Windows as a Service” is part of their design, this will ensure that the team is always up to date on the latest features as well as technologies available for their application.
2. **Operating Systems Architectures**: For their systems architecture I would have to suggest that they go with serverless with AWS S3. This not only shifts the burden of upkeeping and maintaining to the provider but it also reduces overhead and cost since it is a per-instance cost. Now, as the application grows, they might pivot away from this model but for now it is the most ideal option.
3. **Storage Management**: Staying true to my previous suggestion, I would suggest the team to continue using a third party provider, in this instance the clear winner would have to be once again Amazon S3 Cloud Object Storage due to its pricing model, long standing record, as well as reliability.
4. **Memory Management**: Windows memory management offers virtual memory for devices operating virtually that are then mapped by the kernel and user mode components in the physical systems. The physical memory is dependent on what RAM we currently have in our systems, and the virtual memory is dependent on the approximate 16TB that Windows currently supports. All this to say that we will be using serverless via AWS Lambda which eases this need to invest heavily in our own servers for now. Regardless, the virtual memory would be used to run the applications and users in our systems, and the physical memory to handle these operations.
5. **Distributed Systems and Networks**: For our distributed systems, we will be implementing micro-services due to its ability to break up segments of the work into smaller chunks, lower time to launch as well as the ability to future implementations to work seamlessly. It would also allow for the team to not have to worry about all of their services going down at the same time if one of their services fail.
6. **Security**: Windows Defender is a pretty standard practice but due to us operating publicly we would want to have regular security audits, as well as a team that is capable of being proactive by analyzing the latest trends and vulnerabilities such as DDOS attacks that are currently happening in the space or viruses targeted at small operations seeking a quick payout. Something I’ve always thought was very clever is bounty hunter events where you pay individuals to hunt for bugs or even offer a closed beta test for every launch.