

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/21/2024 | Justin Faircloth | Completed executive summary and design constraints |

## [Executive Summary](#_heading=h.35nkun2)

We have been tasked with developing a web-based game that is based on the client’s game, Draw It or Lose It. Currently, it is only an Android app and they would like us to create a web-based version that can run on multiple different browsers. To do this, we will be creating the game in Java and making use of its object-oriented programming. We will use different classes with the appropriate methods and attributes and special patterns, such as the Singleton pattern, to recreate the rules and flow of the game and meet the requirements of the client.

## 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](#_heading=h.1ksv4uv)

* We must accurately implement the rules and features of the game, such as the ability to have one or more teams, multiple players per team, the ability to name teams, etc. The game must function properly according to its rules.
* We must use Java and make use of its object-oriented programming design.
* The game must be built in a web-based environment.
* It must be able to work on different browsers, such as Google, Safari, etc.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

The UML diagram below details the different classes and how they interact to make the entire application function. The classes are Entity, Game, Team, and Player, where Entity is the base class with the others inheriting from it. Game aggregates Team, showing that the game can have one or more teams and Team aggregates Player, showing that a team can have multiple players.

The Game class lets you create a new game and add teams to the game and hold the teams in the game. The Team class lets you create new teams, add players to them, and holds information on the players on the team. The Player class allows the creation of new players.

The UML displays inheritance so that less code is written and properties are shared between the base class and the classes inheriting from it. Encapsulation is used to make sure that the game is managed within the classes and relevant information and functions are only showed when necessary.

**"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](#_heading=h.z337ya)

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 to run servers. However, a lot of storage is needed to do so and macs are expensive to just use as a server. | Most web hosting servers use Linux. It isn’t very resource intensive, is flexible, highly compatible with other systems, and there’s plenty of documentation. You also don’t have to have a license | You can use Windows to host servers, as well. They are pretty secure and easily set up. You do have to buy the license to host servers, which can be expensive. There are many security features | Possible, but not advised. Mobile devices usually lack the memory and power to run servers. Lower-intensity servers may run fine but bigger servers should use other devices. |
| **Client Side** | Macs are very popular with programmers. While expensive, they are secure and easy to use. SwiftUI is easy on the eyes and it is overall very stable. The main concern would be the price of the machine. Also, mac apps can only be developed on a Mac so you would need someone who is used to using Swift | Linux is very popular as well. It is free and well-documented and open-source, so finding someone skilled with using it would be easy. Linux seems to take a long time to develop apps. | Windows licenses are expensive and perhaps the most complicated to develop projects on. Developers will have to be especially familiar with Windows to make full use of it | Mobile device programs are much different so developers with experience in this realm are needed. They must be familiar with accounting for the unique aspects of mobile, such as touch. |
| **Development Tools** | Mac has xCode, which is Apple’s IDE for developing Mac apps. Mac apps are written using Swift | Linux apps are written mostly with Java and Python. Python is usually installed automatically. VIM and Geany are a couple of highly recommended IDE’s for Linux. | C++ seems to be the most prominent language for Windows programs. Visual Studio is a VERY popular IDE for windows and so is Eclipse. | Ios apps must be developed on Macs, so the Mac information applies here. Android apps use mostly Kotlin and Java. the Android software development kit (SDK) is used to develop apps |

## 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**: I recommend hosting the app on Linux servers. It’s free, has good documentation, and is very secure. On top of that, it is extremely compatible with other platforms which would make it easier to expand to new platforms.
2. **Operating Systems Architectures**: Linux uses the Linux kernel, which is extremely efficient and flexible. The architecture is modular and will lend itself towards efficient management of program resources. In addition, it supports multi-user capabilities. With all of this in mind, it is perfect for the Draw It or Lose It app.
3. **Storage Management**: It would be most cost-effective and logical to purchase cloud storage, so we will not have to maintain our own physical storage. Amazon Web Services S3 is a popular and well-maintained cloud storage service but can be a little pricey. Wasabi is a lower cost alternative. Either would allow us to purchase only as much storage as we need.
4. **Memory Management**: Linux overall has efficient memory management. To supplement that, there are multiple ways we can use this operation platform to ensure quick and efficient memory management. First, we can use the JPG format for the pictures as it takes up less data and can be loaded quickly. We can also make use of image compression, which will also take up less space until we need the photos. We can also make use of image caching where it is needed.
5. **Distributed Systems and Networks**: The game’s distributed system will include the cloud storage, the Linux-hosted app, and the client-side devices, such as PC or mobile devices. To facilitate communication between the server and the client devices, RESTful API will be used. The user’s network speed will affect the performance of the app, though. To account for network hiccups or crashes, we can also have a backup system to save the current user’s game status for the duration of their session.
6. **Security**: Linux is already a very secure platform, so we don’t have to worry too much about threats from their architecture. In addition, we will authenticate users and make use of RBAC for the program. The different roles within the program will be separated and lower the chance of code being altered or information being accessed.