

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## 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 4**

[**Evaluation**](#_Toc115077325) **5**

**Recommendations 6**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/09/23 | Christopher Carnell | Initial Design |
| 1.1 | 07/29/23 | Christopher Carnell | Updated development requirement and recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to expand their current game *Draw It or Lose It* to a web-based game on multiple platforms. They will require a singleton pattern for the game instance, a list for the games each with unique game IDs, player IDs, and team IDs. A list for each team with unique team ID and team name, with an associated list of players on the team with unique names.

## Requirements

-Multiplatform: Android, IOS, Web

-Multiple teams per game

-Teams with multiple players

-Unique game and team names

-Only one instance of game

## [Design Constraints](#_2et92p0)

Based on the prompt there were no business constraints I could see such as budget or timeframe.

Cross platform app development, allows a single codebase to run on multiple platforms. There are a few courses of action. If the goal is to have an actual app on Android and iOS stores, a cross platform language such as Flutter should be used to speed up development while keeping costs low, however if they just want a website that can be opened from the browser on any device Java will work fine. Android uses Java or Kotlin while iOS is limited to Swift or C only. There are options to convert Java to C such as Codename One but its subscription based.

A Linux webserver running on Amazon Web Services is the most cost-effective solution with scalability. Likely a SQL server as well to keep track of leader boards, players, and games. AWS has a generous free tier so if there are no players resources won’t be used resulting in no cost. On the other hand, if there are millions of players the number of servers can be increased to handle the increased traffic.

## [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 object-oriented programming principles demonstrated in the diagram are Inheritance, encapsulation, and abstraction. Inheritance is used by using the Entity class with an id and name attribute that Game, Team, and Player inherit from, which allowing us to reuse the same code. Encapsulation is displayed by using private variables and functions. It displays abstraction by limiting what each class can access in the other classes.

The UML diagram below is broken down into seven classes. The ProgramDriver class has the main function which runs the program. The SingletonTester class has one public function to test the Singleton in GameService. The GameService class contains attributes for a list of Games, longs for the GameId, PlayerId, and TeamId, and a Singleton instance for the GameService. GameService also contains public functions to get a copy of GameService, add a new game name, get the game Id, get the game name, get the number of games, get the player Id, and get the Team Id. There is a private constructor for the singleton.

The Game class has a list attribute for the Teams and public functions to set the game, add a team, and convert the team data to a string. It also inherits data from the Entity class including the Id and Name.

The Team class contains a player list attribute with public functions to set the team id and name, to add players to the team, and convert the team data to string. It also inherits data from the Entity class including the Id and Name.

The Player class has no attributes and two public functions to set the player Id and name, and convert the player data to string. It also inherits data from the Entity class including the Id and Name.

The Entity class is the base class for Game, Team, and Player and has public functions to set the id and name, retrieve the id or name, and convert the entity data to a string. There is a private constructor function as well.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Characteristics  Unix based  GUI  Advantages  Performance  Stability  Security  Weaknesses  Cost  Limited hardware choices  Flexibility  Uncommon  Licensing | Characteristics  Unix-like  Open Source  Command Line  Versatility  Advantages  Free  Customizable  Performance  Stability  Excellent web hosting  Weaknesses  Learning Curve  Software compatibility  Driver Support  Limited Support | Characteristics  GUI  Compatibility  .Net framework  MS IIS  Advantages  Ease of use  Compatibility  Support  .NET and other MS technologies  Powershell  Weaknesses  Cost  Resource usage  Security  Performance | Characteristics  Multiple OS  Advantages  Portability  Battery  GUI  Weaknesses  Limited Resources  Connectivity  OS restrictions  Power  Heat  Security  Cost |
| **Client Side** | Hardware Cost  Licensing Fees  Learning Curve  macOS Knowledge  Language Knowledge  API Knowledge  UX Design | Free  Learning Curve  Linux Knowledge  Open source  UI Consistency  Compatibility | Hardware Cost  Low Software Cost/Free  Licensing fees  Windows knowledge  UI Design | Hardware Cost/multiple devices  Licensing fees  Distribution fees  Learning Curve  Frequent Updates  Mobile development knowledge  UX Design  API knowledge |
| **Development Tools** | Languages:  Swift  Objective-C  C/C++  Java  Python  IDE:  Xcode  Other Tools:  Homebrew  Git  CocoaPods | Languages:  C/C++  Python  Java  IDE:  Eclipse  JetBrains  VS Code  NetBeans  Other Tools:  GNU Compiler  GDB Debugger  Git  Docker  Yum | Languages:  C#  C/C++  Visual Basic  Python  Java  IDE:  Visual Studio  JetBrains  Eclipse  Other Tools:  .NET Framework  MSBuild  NuGet  Git  PowerShell | Languages:  Swift  Objective-C  Java  Kotlin  Dart  IDE:  Xcode  Android Studio  Visual Studio  JetBrains  Other Tools:  Gradle  CocoaPods  Expo  Flutter  Package managers |

## Recommendations

1. **Operating Platform**: The server should be run on the cloud and Linux based. This is the most cost-effective way, it’s scalable, reliable, and secure.
2. **Operating Systems Architectures**: Cloud hosted using authoritative Linux servers. Linux servers are the cheapest option and offer the best web hosting. Linux will use the least system resources allowing for more to run the actual game instance.
3. **Storage Management**: All the cloud hosting platforms like AWS and Azure offer free tier storage which will most likely accommodate the image storage and future expansions before any cost is incurred. Relational databases should be used to store things like player data and high scores, as well as links to the image storage locations.
4. **Memory Management**: The server only needs enough memory to run the OS, and instances of the game. The images do not need to be stored in memory, or even on the server itself. By breaking the images down into smaller pieces, they can be streamed over the network faster than full size images. Each platform will only have to load one image at a time so around 8mb max for the image, then we rely on garbage collection to clear it from memory before we load the next image.
5. **Distributed Systems and Networks**: Since the game is web based and the server handles all the logic a cloud-based service like AWS, Azure, or Google Cloud should be used. Communication between the devices and server will be done using TCP/IP connections. There should be one server that all devices initially connect to which handles login and load balancing. After the user is authenticated the load balancing server directs the client to an available game server. HTTPS will be how the data is sent back and forth.
6. **Security**: Data should be sent encrypted using HTTPS and all passwords should be stored and hashed at a minimum in the database.