

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 |
| --- | --- | --- | --- |
| 2.0 | 4/20/2025 | Kai Grignon | Improving the evaluation and recommendation sections |

**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 client has a game that is currently available on Android devices, and they wish to expand their customer base by offering a web-based version of the game. The game involves teams of players trying to guess the meaning of an image that is gradually revealed.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* One difficulty will be allowing different people to play together on different devices. While the Android app presumably had multiple people playing on the same device, people will want to play remotely with each other now that the app is in web form. One solution would be to have an auto-matching algorithm that matches people with other players at a similar skill level. However, in some cases people will want to play with their friends specifically. Solutions could include letting users share unique codes or “join game” URLs in order to play with their friends.
* Some players will access the game from mobile browsers with small screens, while others will use tablets, laptops, and desktops. This variety of screen shapes and sizes may make it difficult for some players to view the image. One possible solution is to allow the players to freely zoom in and out on the image.

## [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 parent of Game, Team, and Player. It defines the id and name properties that are common to its child classes, as well as a basic constructor that sets id and name and several getter methods. The default constructor is kept private for encapsulation, so that no instance of Entity can be create without a name and id.

The Player class is a very thin wrapper around the Entity class. The only new functionality is the overridden constructor and the overridden toString() method. The use of inheritance here allows for a slight modification of Entity’s functionality without the need to re-write all the code.

The main feature of the Team class is the list of players that it contains. This list is encapsulated so that it can deal with invalid input and prevent players with duplicate names or IDs to be on the same team. Like Player, Team inherits the id and string properties and the associated methods, while it overrides the constructor and toString().

In turn, Game contains a list of Teams, similarly encapsulated to prevent duplicates. Again, it inherits name and id, while overriding constructor and toString().

GameService contains a list of Games, and many of its methods are geared towards auto-assigning IDs to new games as they are added, as well as retrieving games by name or ID. It does not have its own name or id, however, since it follows the singleton design pattern, meaning that only on instance of GameService can exist.

GameService is used by SingletonTester (although this relationship is not shown on the diagram) and SingletonTester is used by ProgramDriver.

**"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** | Mac’s main advantage is its intuitive user interface, which will not be used regularly on the server side. Additionally, Mac’s prohibitively high hardware prices make it a poor choice for a server-side operating system. | Linux is free to use, and it can be stripped of its user interface for efficiency and run on almost any hardware. However, it is not backed by a large company, so it may be more error-prone and less secure. | Windows does have a specific OS for server-side operations, which has been professionally tested and works out of the box. However, getting a license is fairly expensive. Despite this, I would recommend windows server as the best OS to use for the server side. | The many hardware and software limitations of mobile devices make them a poor choice for a server-side system. |
| **Client Side** | Since we are developing a web application, the client’s operating system does not matter as much. However, it is important to note that many Mac users use Safari and/or Google Chrome to browse the web. The game must be tested to work on these browsers. | Linux users may use many different and less well-known browsers. While it may be impossible to test the game on all browsers, it is important to at least test it on the most common browser engines, like chromium and webkit. | Windows users often use Microsoft edge and/or firefox to browse the internet. | Even within the context of a browser, the unique screen shapes and sizes of mobile devices will make the game development challenging. It will be important to make the web app responsive to different screen sizes. |
| **Development Tools** | For the web application, HTML, CSS, and JS could be used, or a framework like React or Django. If the game were made into a native Mac app, the Swift language would provide the most native integration, while another language such as java would allow much of the code to be reused between different platforms | For web app, requirements are same as Mac. For a native app, many languages, including Java, C++, and Python could be used. | For web app, requirements are same as Mac. While a native app could be developed in many languages, the .NET framework and C# are both made for windows app development. | For web app, requirements are same as Mac, although using a framework like Bootstrap could help with different screen sizes. For a native app, Java/Kotlin is the default for Android, while Swift is made for IOS. On the other hand, there are frameworks that work with both Oss, like React native and Kotlin multiplatform. |

## 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**: Since the game is already available on Android and the web-based version will work for most laptop and desktop players, IOS devices should be the next platform to expand to. While Android is more popular than IOS, almost a third of the mobile phone market share belongs to IOS1. The web application will be available for all users, including on IOS devices. However, a native application for IOS can leverage persistent local storage and system APIs and can be tailored specifically to mobile screens to create a “smoother” experience that will attract more players to the game.
2. **Operating Systems Architectures**: IOS has a layered architecture, with four main layers: the core OS, core services/APIs, multimedia, and Cocoa Touch for the UI2. The game will primarily be built on the Cocoa Touch layer, but it may require some interaction with the core services. For instance, there could be an option that allows the user to take a profile picture or choose it from their gallery. This would require using the camera/gallery API.
3. **Storage Management**: Cloud storage would be most appropriate, since it allows users to sign in and play on different devices. See distributed systems below. However, there could also be an offline mode where some or all of the pictures are stored locally on the device to allow users to play when the device cannot connect to the server.
4. **Memory Management**: Assuming that the Swift language is used to develop the IOS app, memory management will not be a concern. Swift’s automatic reference counting system will take care of memory management, eliminating the need for extreme care around memory leaks, etc. Best practices should still be used for efficiency.
5. **Distributed Systems and Networks**: A cloud computing platform such as AWS or Google Cloud will allow users to store their information on the cloud, so that it can be accessed wherever and whenever they sign in. It will also act as a server to keep track of games in progress, allowing people to join teams and play games with their friends, even when they are in different locations.

The backend server currently being developed for the web application can be extended to work with the new IOS app without too much trouble.

1. **Security**: The best security policy is for the application to collect as little information from users as possible, so as not to expose users’ information. Other security measures should include end-to-end encryption and two factor authentication (for administrators, at least). All sensitive information that is stored locally on the device or on the server should be encrypted, and all communications between device and server should use the secure HTTPS protocol.

1statcounter. (2025, Mar). *Mobile Operating System Market Share Worldwide.* <https://gs.statcounter.com/os-market-share/mobile/worldwide>

2GeeksForGeeks. (2024, Aug 23). *Architecture of IOS Operating System.* <https://www.geeksforgeeks.org/architecture-of-ios-operating-system/>