

Draw It Or Lose It

# **CS 230 Project Software Design Template**

Version 13.37

## 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 | 9/11/23 | Alex Frankel | Completed Executive Summary, Requirements, Design Constraints, and Domain Model |
| 2.0 | 9/30/23 | Alex Frankel | Completed Evaluation of different operating platforms |
| 13.37 | 9/15/23 | Alex Frankel | Provided Final Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

One instance of the Game service will be active at any given time. This service will include zero to many individual games, teams, and players. This can be accomplished by instantiating the game service as a ‘singleton’, which in itself will contain information related to current games (as an ArrayList) as well as the ability to add new or find existing games. These games will contain a list of teams and within these teams lists of players. The GameService will be accessed by a static reference which means it is consistent through all users which access it.

Android’s native application programming language is Java, so any existing application on Android can potentially be migrated to a web-based Java platform. This, in turn, will allow for usage on multiple different platforms.

## Requirements

*-Web-based game server that serves multiple platforms*

*-A game will have the ability to have one or more teams involved.*

*-Each team will have multiple player 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)

A web-based application can be coded in Java. While Java can be utilized on many different operating platforms, there is a tradeoff with performance due to it’s high-level nature. There will be a need for hosting the game server, this can be done by renting or buying a dedicated server. This need will change based on concurrent users, requiring more bandwidth for more users. Luckily, it seems like the game itself is not very resource-taxing. The game’s user interface, which will begin development at a later date, can be created using GUI packages included with Java.

## [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)

We start by explaining the Entity base class. Entity describes a non-specific member of our game service. Games, Teams, and Players inherit basic features from Entity and have an is-a relationship (they are all Entities themselves). An Entity has an id and a name. These fields are inaccessible to other parts of the program and must be accessed through calls getName and getId functions. This principle is known as encapsulation, where internal parts of a program are protected from system-wide access. An Entity can also be displayed by a toString method that allows it to be represented textually with its id and name. All the children of Entity will also have name, id, and will override the toString so that it reflects their individual identities. The GameService can only be accessed by reference by each Game, Team, or Player. There can only be one instance of the GameService, but there can be an arbitrary amount of Games, Teams, and Players. This instance is accessed by the getInstance method. The GameService has an aggregate relationship with Games with a multiplicity of 0 to many. This means that for the instance of the GameService, there can be a minimum of 0 games to many games… for a given GameService this is called an ‘optional’ relationship. For a given Game there can be 0 to many Teams, and for a given Team there can be 0 to many Players. The ProgramDriver runs the program and instantiates these Games, Teams, and Players and displays them to output. SingletonTester does the same thing and tests whether the GameService has a single static instance (meaning that it is the same throughout all programs that use it).

**"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** | The built in web server for Mac is called Apache. With Apache you can host HTTP servers from your own IP. However, unique domain names must be purchased from sites such as GoDaddy or HostGator. They offer cloud services as well. | Apache can also be used to host servers on Linux. Cloud based services may be the best decision for hosting a web app. | Windows Server bundles services like Microsoft Azure which with a subscription can host web servers on the cloud. Cloud servers can be scaled up in size according to need. Apache servers can be run on Windows as well. | Server hosting should be done on a dedicated machine with enough resources for its use. Mobile devices can access a web app through the browser, and webpages can be easily converted into app store ready applications. |
| **Client Side** | Mac OS’s default browser is Safari. Mac can use all sorts of different 3rd party browsers. A web-based application need only be accessed through a web browser and, based on system properties, change accordingly (IE the display/resolution). See column 3. | Linux has access to all sorts of web browsers.  A web-based application need only be accessed through a web browser and, based on system properties, change accordingly (IE the display/resolution). See column 3. | **Maven, which we have used to build this project so far, provides support for the use of profiles based on OS and system properties that can change the way the application functions in order to make up for these differences.**  **THIS APPLIES TO ALL PLATFORMS and can help make apps compatible with all platforms.** | Mobile devices have access to the most popular internet browsers, which would be used to access the game. Websites can be accessed through Mobile OS-specific applications and listed on their respective app stores. This will be the most reliable way of achieving compatibility and can be done cheaply, quickly, and easily. |
| **Development Tools** | For Mac OS applications, the main IDE provided by Apple is Xcode. Xcode supports many languages but the preferred ones are traditionally Objective-C, which is based on C, and Swift which was created more recently for app development. Xcode has interactive UI creation tools to make the process easier. The Apple Developer Program costs $99 a year and provides more powerful tools and app distribution | Linux can run most IDE’s including Eclipse, the one we are currently using for the development of the game. | React.js allows for integration of javascript and html to create web based UI. It supports HTTP communications.  Windows can be used for most programming languages, provided their source code is downloaded on a given system. Microsoft’s kernel was developed mostly in C, making it and its successors (especially C++) a preferred language due to its high power but also its versatility. | See Mac Development Tools section, it applies to mobile as well as traditional machines.  The Google Play store can host mobile app versions of any website. There is a one-time fee of $25 to publish Android apps. Java was the original native language of Android apps, which is good because it can be run on any operating system. It has since then been replaced by Kotlin. Today, that is Google’s preferred language. |

## 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 the Java platform since it is a Virtual Machine that can be run on any operating system. It can be ported to mobile devices by porting the app to a webpage that can then be accessed through mobile apps on iOS and Android.
2. **Operating Systems Architectures**: Java, like previously stated, runs on a platform-independent bytecode that has been adapted to many different operating systems. This way, though the base architecture may differ, Java programs can run independently of their underlying operating systems. Apache Maven can be applied to our Java project and allows for differing profiles based on OS and screen resolution (this helps scale the app for compatibility reasons).
3. **Storage Management**: Draw it Or Lose It is a relatively compact application. It only contains roughly 1.6GB total of images. This can be stored practically for free on any cloud service. Specifically, if we are to scale up our storage needs based on an increasing player base, we may need more storage for each user’s game data (if we decide to implement that). I would recommend Amazon Web Services for scalable storage solutions, but it probably won’t be necessary anyway since game data will be very compact.
4. **Memory Management**: Applications must utilize memory management in order to prevent memory leaks that can harm user’s machines. Java does this pretty much automatically. It is an industry standard that however this application is coded that memory is managed properly. User’s machines will not need to worry about memory usage because all they need to render is a webpage with basic UI such as menus as well as a few different pictures for each round in the game. Using a concept such as ‘lazy loading’ would mean that only the images necessary for gameplay will ever be loaded into memory, which is a total of about 32MB for the images at any given time.
5. **Distributed Systems and Networks**: Every user interaction can be transmitted using the client-server model. AWS can host websites and is scalable for performance needs. We can use asynchronous functions such as listeners for user input and then communicate their entries to the game server using HTTP methods. The client need only access the webpage to render game data and accept input, the server can handle all of the heavy lifting and computations.
6. **Security**: The obvious solution would be to encrypt all stored user data such as passwords. However, this is an incomplete idea. For example, a user may make use of an HTTP ‘PUSH’ function in order to send a username and password request to a server. The client need not compare the login data directly, but by making the comparison on the backend it can eliminate any holes in security caused by saving user data in the client. There will be no way for a user to obtain this information. If the user is valid, the website will simply redirect to the game. On top of that, two-step authentication involving an email or mobile device will protect even users that have exposed their password! Using AWS to host the game would come with user authentication built in.