23

The Gaming Room

# **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 |
| --- | --- | --- | --- |
| 1.0 | 1/17/2024 | Jaden Miller | Edits added to Executive Summary, Requirements, and Design Constraints |
| 1.1 | 2/22/2024 | Jaden Miller | Edits made to Domain Model |
| 1.2 | 3/3/2024 | Jaden Miller | Edits made to Evaluation |
| 1.3 | 3/3/2024 | Jaden Miller | Edits made to Recommendations |

**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)

A new client, The Gaming Room, is requesting a web based game serving multiple platforms, relating to their current game, Draw It or Lose It. Among their requests, they would like the game to involve more than one team and have multiple players assigned to each team. Along with this, they requested that each game and team name be unique in order for users to avoid using existing names for teams, and then only one instance of the game may exist in memory at any given time.

## 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)

While creating this application, there will be a few constraints that may limit our work that includes:

* Because the game is web-based, the code will need to be read within a browser, meaning an HTML application may be necessary.
* Since multiple players will be assigned to a team, one or more lists will need to be made
* With a web-based application being assigned, a smaller team may need to be assigned, as it will be a fairly small application.
* Only one instance can exist in memory, meaning a singleton pattern will need to be implemented.
* Team names need to be checked for existing names, meaning an iterator pattern will likely be necessary.

## [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)

**In the UML Diagram below, the code starts with a main Entity class inheriting to four separate classes. This class contains two calls for a game ID and name, along with a call to the class itself. Next to this class, a singleton testing class is implemented to ensure that the used singleton pattern is in working order. Below, each class relies on the previous. For example, the GameService is the class is needed by the game class in order to add games, hence the addGame public calls, which can be multiple due the zero to many multiplicity. The Team class needs the game class in order to add teams, since the game class contains the addTeam public call, also allowing multiple calls with the zero to many multiplicity. Finally, the Player class requires the team class to add a player, as it contain the public addPlayer call, again allowing multiple calls.**

**"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 has OS X Server available for use, however obtaining hosts will be difficult with high costs.  OS X Server: $499 USD for up to ten clients; $999 for Premium | Linux is open source, allowing for lower maintenance and licensing costs.  Serves as most popular webhosting OS.  Cloud storage providers such as  Google recommend Linux over other OS’s | GUI Based.  Tend to have higher licensing costs.  Costs range from as low as $500 to as much as $6,200 for higher level offices/workspaces.  More limited hosting platforms. | Not readily prepared for serving more than one user.  Usually has more limited hardware capabilities with being mobile.  Unknown costs for hosting with need for scratch made tools. |
| **Client Side** | Mac OS needs to be running latest Xcode version.  macOS SDK runs in not well known languages such as Objective-C and SWIFT.  Smaller chance for profit and market growth with only 16% usage of the market. | Usually a simple development process, with options such as Python, Java, C/C++, etc. available.  Support for more than one user is also available through Linux | Usually developed using C# or .NET;  Serves as a platform for more than one user since XP  75% of computer users actually prefer Windows over other OS’s, providing a better market overall. | While not designed for more than one user, development for iOS and Raspberry Pi is fairly simple.  May be able to take code developed from a PC based OS such as Windows or Linux.  Similar requirements to Mac for iOS, as well as for Windows with Raspberry Pi. |
| **Development Tools** | Objective-C and SWIFT as languages  Xcode is most common Mac IDE.  $99/year for XCode per developer. | Usually developed with more common languages like Java, Python, C/C++.  Free IDEs, including NotePad++, PyCharm.  Eclipse not available on Linux but can read and write all abovementioned languages | Mainly developed in C# and .NET.  IDE’s such as Eclipse, Microsoft Visual Studio.  Visual Studio can run anywhere from $45 - $250/year, based off of selected features. | Raspberry Pi is Java based using Android SDK, developed by Google and available in an application known as Android Studio, which is free.  For iOS, very similar to Mac OS, developed within SWIFT and Objective-C in Xcode.  $99/year for XCode per developer. |

## 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**: Using an open source platform such as Linux would likely be the go to move in this situation. With the lower costs for licensing, along with no limit to data center access, it would allow for more opportunities. Linux also has an agnostic frontend, allowing for a wider range of available languages to write in. Finally, although it is open source, Linux serves as a secure and safe OS for developers.
2. **Operating Systems Architectures**: The chosen architectures would serve for backend servers managing the game environment along with client-based and front-end rendering. Low latency is not necessary between the frontend and backend of the game, since it does not need a twitch reaction. Something newer for a backend would allow for more scalability within the program, though the provider of the cloud storage would need to be decided. For frontend, using it for rendering supplies an offload of heavy supplies/materials, allowing for lower data costs overall.
3. **Storage Management**: No specific data assignment needs to be made unless the game owner wants do so. Utilizing an SSD or even an HDD would suffice perfectly.
4. **Memory Management**: Within Linux, Pagecache is used for data stored within main and virtual memory for any allocated pages. Demand paging is used to allow for less memory usage as non actively used ages are not loaded. Android Runtime, or ART, along with Dalnik use paging and memory-mapping for memory management, meaning any memory edited by the application will remain still within a system’s Ram. For iOS, memory management supports ARC, or Automatic Reference Counting. Because of this, objects do not have to be kept and therefore the development application, Xcode, handles it.
5. **Distributed Systems and Networks**: For distribution, both the frontend and backend will be responsible for communication through RESTful APIs. The RESTful API usage will allow for more client/server transparency to the chosen platform.
6. **Security**: The security area will likely need Role-based authorization, with the interface needing developed to create effective security and administration. There will not be any allowed administrators other than the developers of the application, and API’s will use various layers of protection, including encryption, 128-bit keys, etc. Along with this, there will be a firewall added to ensure another layer of protection for the server.