

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/23/2022 | P D Pattison | Project One |
| 2.0 | 02/01/2022 | P D Pattison | Project Two |
| 3.0 | 02/19/2022 | P D Pattison | Project Three |

**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 Game Room is looking to have a web-based game developed that is based on their current game Draw It or Lose It. Draw It or Lose It is loosely based on the 1980s television show called *Win, Lose or Draw*. Unlike on the television show where a player would draw clues to get their team to guess the puzzle, the application will render drawings from a large library of stock drawings. A game will consist of four rounds that last one minute each. Drawings will slowly be rendered taking 30 seconds to fully render, and if a team is unable to guess the puzzle within the 30 second render time, the opposing team will be given 15 seconds to offer one guess for the puzzle.

## [Design Constraints](#_2et92p0)

The game should be web-based to allow players on multiple platforms to play. A game should only allow one instance to exist in memory at a time. This will require the Game class to be a singleton class. Game and Team names should be unique. This will require a check of existing Game and Team names to verify that no duplicates are used. A game will have one or more teams, and each team will have multiple players. This will require the Game class to have a one to many relationship with the Team class, and the Team class to have a one to many relationship to the Player class.

## [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)

GameService class should be a singleton class which will guarantee that only one instance is allowed to exist at a time. The GameService class instance will contain zero or more Game class instances. A Game class Instance will contain zero or more Team class instances. A Team class instance will contain zero or more Player class instances. Game class, Team class, and Player class will all inherit from the Entity class.

**"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** | MacOS is based on Linux, so even if out of the box Macs don’t have a web server installed, installing one would be easy enough. Usually more used as a desktop operating system rather than a server operating system. MacOS does require a license for the OS, however as it is based on Linux, any required server software would be open source and free. | Linux is the quintessence of server operating systems. It can be a little tricky to setup, however Linux makes a very powerful server platform. Linux is also open source and free. | Windows server platforms are quite powerful. Windows is closed source and proprietary. Licensing Fees can be quite expensive. In addition the Internet Information Server is proprietary and expensive. Apache Web Server is Open Source and does run on Windows. | As far as the server side goes, Mobile Platforms are not up to the task. |
| **Client Side** | Client side application would just be a web browser. Client side would be developed and deployed on the server, and served to the clients through a web server. | Client side application would just be a web browser. Client side would be developed and deployed on the server, and served to the clients through a web server. | Client side application would just be a web browser. Client side would be developed and deployed on the server, and served to the clients through a web server. | Client side application would just be a web browser. Client side would be developed and deployed on the server, and served to the clients through a web server. |
| **Development Tools** | The Java Language, along with the DropWizard Library would be ideal for this application development. An excellent IDE for Java that is usable on Mac, Linux, and Windows is Apache NetBeans. Java, DropWizard, and NetBeans do not require any additional licensing. | The Java Language, along with the DropWizard Library would be ideal for this application development. An excellent IDE for Java that is usable on Mac, Linux, and Windows is Apache NetBeans. Java, DropWizard, and NetBeans do not require any additional licensing. | The Java Language, along with the DropWizard Library would be ideal for this application development. An excellent IDE for Java that is usable on Mac, Linux, and Windows is Apache NetBeans. Java, DropWizard, and NetBeans do not require any additional licensing. | Mobile platforms do not make good development environments. |

## 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 that the server for the game should run Linux.
2. **Operating Systems Architectures**: Linux is diverse enough that you can get a version of Linux to run on any hardware. In addition, Linux is free making it a lower investment requirement to get setup. Finally, Linux has all the server software available for any client needs.
3. **Storage Management**: The storage space requirement for the 200 images is only 1.67 gigabytes. The application requirements are not very high. It is recommended that the server implement a Redundant Array of Independent Disks (RAID) Array for storage in order to minimize the loss should a hardware failure occur.
4. **Memory Management**: The requirements of the application are to reveal a prerendered image steadily during duration of the round. To smoothly accomplish this, it is recommended that the application be designed such that the image for a round is fully loaded into memory. Reads from memory are much faster than reads from a disk, and having the image fully loaded into memory will eliminate the lag from reading from disk. It is also recommended that after each round the previous rounds image be fully unloaded prior to loading the next image.
5. **Distributed Systems and Networks**: The easiest way to accomplish having the application be distributed and allow for interaction with any client operating platform, it is recommended that the application be designed as a REST API using web browsers on the clients side. This will greatly reduce the development requirements as the client side will just be a standard web browser. Network outages would be catastrophic, and should be mitigated as much as possible. To help mitigate the impact a network outage would have, it is recommended that the servers be hosted in a data center with redundant network connections, and high up times.
6. **Security**: Linux is, or at least can be, a very secure operating system. It is recommended that the server be hardened for security as much as possible. REST applications can be designed to use user authentication. They can also be designed for users to have roles. It is recommended that roles be implemented in order to separate roles, which will increase the overall system security. Finally, it is recommended that no passwords be sent in plain text over the internet.