

The Gaming Room

Gregory Greene

# **CS 230 Project Software Design Template**

Version 1.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 06/13/2022 | Gregory Greene | Updated information on The Gaming Room Design |

**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 Gaming Room intends to build a web-based game that will be run on multiple platforms. Currently the client’s game “Draw It or Lose It” is only available on android platforms. This game has multiple teams with several people on each, where the teams spend 4 rounds of 1 minute taking turns. The program pulls a picture from a library of images, as the image is rendered the teams the current team will guess the image. If the team is unable to guess correctly then another team will be given 15 seconds to guess.

The client’s application must consist of one or more teams, must run on multiple platforms (Android, IOS, etc.), and only one instance of the game may exist at a time. Team and Game names must be unique.

## [Design Constraints](#_2et92p0)

* One or more teams
* Each team consists of multiple people
* Team and Game names must be unique
* Only one game instance can exist at a time
* Must run on multiple platforms
* Web server to host client’s application and store images for the game
* Ensure application is secure across all platforms

We must focus on these requirements during development. As we progress, we must ensure the code is compatible with multiple platforms (Windows, Linux, IOS), and can be easily updated when 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)

The Entity class creates a link between the Game, Team, and Player Classes. These classes all inherit from the Entity (super) class. The UML allows us to show the inheritance of items such as name and id visually. We can use UML to see aggregation (Has A) with Team and Player, such as GameService has Games, and Game has a Team. The UML diagrams allow easy visual understanding of the reference an instance has to another 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.**

## Did not fill out below sections as not listed in module 3 rubric. Assuming sections below are for future weeks.

## [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 OS X Server available and inexpensive.  Flexible terminal commands to make changes and access the server. | Less popular so may require a professional to manage the server.  Flexible, and cost efficient | Most common/easily understood.  Microsoft Server/Software can be costly.  Allows more features than other OS platforms | Mobile server could have problems depending on location/travel.  Device specs differ between user’s and could cause problems depending on the device.  Typically, less powerful OS. |
| **Client Side** | Not open sourced and would require expert knowledge.  Similar cost to Windows. | Would require time and expertise, more than other options.  Linux is cost efficient. | Windows is higher cost, like Mac.  Would require expertise and time to setup, possibly less than Linux as Windows is more common. | Hard to implement as there are different OS available across mobile devices.  Potential to be updated more frequently than traditional options. |
| **Development Tools** | Common languages include HTML, CSS, JavaScript, and Swift | Supports Eclipse, Visual Studio, and Atom.  Supports common languages such as HTML, CSS, and JavaScript | Most common, provides access to Eclipse and Visual Studio.  Easy to use for multiple languages such as HTML, CSS, JavaScript, and C# | iOS applications are typically developed in Swift similar to Mac.  iOS and Android allow for common languages such as HTML, CSS, and JavaScript.  Becoming increasingly popular |

## 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**: A cloud-based Linux OS is recommended. Serverless architecture allows for scalability, less maintenance, and lower costs. This also allows for ease of access through various things such as HTTP or API, which provides access and support to multiple environments.
2. **Operating Systems Architectures**: The Linux OS is a modular architecture, where the kernel contains core components, and expands into different modules that contain various services. The modular design improves processing speed and uses less disk space overall. Additional security is provided due to the way Linux uses system libraries for various functions, limiting hardware access.
3. **Storage Management**: A cloud-based system allows for the use of multiple servers to promote virtualization. This allows The Gaming Room to focus on design without much consideration into memory. Linux uses a multi-tiered indexed structure for file storage. This system indexes different identifiers to allow for faster search time of files. This also conserves memory as it avoids using large singular index systems.
4. **Memory Management**: Linux uses virtual memory and demand paging for partial memory execution. This allows multiple programs to run simultaneously and helps reduce memory constraints on the system. Partial memory execution allows applications to run faster which is key for Draw It or Lose It as the developers intend for it to be run across multiple devices with different system specifications.
5. **Distributed Systems and Networks**: A cloud-based, serverless, architecture will allow for Draw It or Lose It to scale as popularity increases. As the application will run across multiple devices and servers load balancing will need to be implemented. Load balancing can be used to provide optimal server usage to the client which provides faster response time and a smoother experience for the user. We can implement REST API exchanges to support communication between client and server across HTTP. If problems with a server occur a distributed system with load balancing can redirect clients to working servers to prevent downtime and other issues. This will improve the user experience and load times overall.
6. **Security**: For this type of application, we will use authorization and authentication to protect user data. Login credentials will provide authentication, while authorization can be provided through role-based access. Users will be assigned roles that will grant permission to edit their username, team names, and play the game, but not the ability to modify the application or images in the game. REST will allow for stateless interaction between the clients and servers which will use client-side cached identifiers. Information such as usernames and passwords can be stored in a hashed database and can be encrypted for further protection.