

Draw It or Lose It

# **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 | 09/18/22 | Rebekah Jackson | * Added Executive summary * Added Design Constraints * Added Domain model text description * Made initial recommendations |
| 2.0 | 10/02/22 | Rebekah Jackson | * Filled in Evaluation table * Removed Instructions from document |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room would like to create a web-based application of their game “Draw It or Lose It” which is currently only available on Android. They would like to create a web-based version of the game to be able to reach more player across different platforms.

## [Design Constraints](#_2et92p0)

* 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.

## [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)

This is a UML diagram of the package com.gamingroom. The Game, Team, and Player classes all *inherit* from the Entity class, which is a general high-level class that holds basic information about the object. This also allows for Polymorphism at runtime since these three classes have an “is a” relationship to the Entity class. Encapsulation is present in the Entity class because the attributes are private, and accessor methods are used to manipulate them as needed. There are “zero to many” associative relationships between the GameService to Game, Game to Team, and Team to Player classes. For each relationship the class on the left “has a” of the class on the right. There is Polymorphism with the toString method being overridden in the Game, team and Player classes. There is also Polymorphism in the GameService class with the use of method overloading. The ProgramDriver class contains the main method, which runs the program. It uses the Singleton tester to make sure that the GameService glass is a singleton.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is a solid choice for the server-side of development. The built in terminal on MacOS makes connecting to and administrating remote systems simple. Similar to Linux, Mac has roots in Unix, so it offers stability and performance that is comparable to Linux. One disadvantage to using Mac is that it requires a device that runs MacOS for development, which tend to cost more than Windows machines. | Linux is a good choice for the server-side development because it is easy to run, offers strong processing power, and is one of the most stable of the choices in terms of server performance. Linux is also one of the lower cost options on the list, and there is a range of Linux operating systems from open-source systems like Ubuntu, to enterprise licensed packages such as Red Hat Enterprise Linux (RHEL). One disadvantage to using Linux is that it can take new developers some time to get used to using a Linux server. | Windows is an okay platform for hosting a web-based software application. They have bundled packages that can provide all of the different software that is needed for developing this type of application. With that being said, Microsoft web servers are known for instability and having issues, and Microsoft support can be difficult to work with. Another disadvantage of Microsoft is the high licensing cost. | A mobile web server hosting platform would not offer powerful performance or stability in operation, as mobile devices are not as powerful as dedicated servers. |
| **Client Side** | Unlike Linux, Mac is proprietary which means that the cost will be higher. Mac requires less experience to work with than Linux, but more than Windows. Also requires a device that runs MacOS for development, which can be costly. | It can take new developers some time to get used to using a Linux server, because the expertise level is higher for Linux based servers. There are a lot of choices when it comes to cost of Linux servers, from free open-source software to packages that require an enterprise license. | Most developers are familiar with Windows and the tools available in Windows, so the expertise level is a little lower. The licensing for Microsoft products can be fairly expensive. | Expertise is a little higher because the developers will need to be familiar with mobile application structure. |
| **Development Tools** | Using IDEs such as Eclipse, IntelliJ/Pycharm or Visual studio the application on this platform could support front end languages such as HTML/CSS and JavaScript as well as backend languages such as Java, C++, or Python. Deployment automation tools like Jenkins could assist in deploy applications to this type of server. | Amazon Web services offers several packages that work with Linux servers. They even offer web based Linux servers to develop with. An AWS package on top of an open-source OS could a lower cost, but well-rounded development solution. | Using IDEs such as Eclipse, IntelliJ/Pycharm or Visual studio the application on this platform could support front end languages such as HTML/CSS and JavaScript as well as backend languages such as Java, C++, or Python. Deployment automation tools like Jenkins could assist in deploy applications to this type of server. | Visual studio/Android studio for Andriod. xCode for IOS. |

## Recommendations

1. **Operating Platform**: I would recommend an AWS virtual Linux based server running AWS services.
2. **Operating Systems Architectures**: Linux is a secure file-based operating system. AWS, Amazon Web Services, is a series of products from Amazon that offer many tools for developers. They offer services such as data management, analytics, server hosting etc.
3. **Storage Management**: I would suggest using the Amazon Simple Storage Service which is a basic scalable storage system.
4. **Memory Management**: AWS has built in services to scale memory management as needed to help reduce memory load as well as costs.
5. **Distributed Systems and Networks**: Cloud based servers handle things like upgrades and outages easily. One way is to have clustered servers that traffic can be routed to in case of an outage.
6. **Security**: AWS has many built in systems that can be used for security. I would recommend basic group-based access in the Linux and web server systems. I also this it is important to use the latest versions of Transport Layer Security when sending and receiving requests to the servers. The code will also need to be written securely to prevent cross-site scripting and other injection type attacks.